FATEC – Anotações da disciplina “Banco de Dados” - Professor : M. Sc. Joilson Cardoso
Ultima atualização 08/10/2018

SUMÁRIO

ANOTAÇÃO 1 Vantagens do Banco de Dados

1.1 Vantagens do Banco de Dados em relação à arquitetura tradicional

1 - Redução ou Eliminação de Redundâncias - Possibilita a eliminação de dados privativos de cada sistema. Os dados, que eventualmente são comuns a mais de um sistema, são compartilhados por eles, permitindo o acesso a uma única informação sendo consultada por vários sistemas.

2 - Eliminação de Inconsistências - Através do armazenamento da informação em um único local com acesso descentralizado e, sendo compartilhada com vários sistemas, os usuários estarão utilizando uma informação confiável. A inconsistência ocorre quando um mesmo campo tem valores diferentes em sistemas diferentes. Exemplo, o estado civil de uma pessoa é solteiro em um sistema e casado em outro. Isto ocorre porque esta pessoa atualizou o campo em um sistema e não o atualizou em outro. Quando o dado é armazenado em um único local e compartilhado pelos sistemas, este problema não ocorre.

3 - Compartilhamento dos Dados - Permite a utilização simultânea e segura de um dado, por mais de uma aplicação ou usuário, independente da operação que esteja sendo realizada. Deve ser observado o processo de atualização concorrente para não gerar erros de processamento (atualizar simultaneamente a mesma coluna do mesmo registro). Os aplicativos tendem a ser multi-usuários.

4 - Restrições de Segurança - Define para cada usuário o nível de acesso a ele concedido (leitura, leitura e gravação ou sem acesso) ao arquivo e/ou campo. Este recurso impede que pessoas não autorizadas utilizem ou atualizem um determinado arquivo ou campo.

5 - Padronização dos Dados - Permite que os campos armazenados na base de dados sejam padronizados segundo um determinado formato de armazenamento (padronização de tabela, conteúdo de compôs, etc) e ao nome de variáveis seguindo critérios padrões preestabelecido pela empresa. Ex. Para o campo "Sexo" somente será permitido armazenamento dos conteúdos "M" ou "F".

6 - Independência dos Dados - Representa a forma física de armazenamento dos dados no Banco de Dados e a recuperação das informações pelos programas de aplicação. Esta recuperação deverá ser totalmente independente da maneira com que os dados estão fisicamente armazenados. Quando um programa retira ou inclui dados o SGBD compacta-os para que haja um menor consumo de espaço no disco. Este conhecimento do formato de armazenamento do campo é totalmente transparente para o usuário. A independência dos dados permite os seguintes recursos:

a - Os programas de aplicação definem apenas os campos que serão utilizados independente da estrutura interna dos arquivos

b - Quando há inclusão de novos campos no arquivo, será feita manutenção apenas nos programas que utilizam esses campos, não sendo necessário mexer nos demais programas. Obs.: Nos sistemas tradicionais este tipo de operação requer a alteração no lay-out de todos os programas do sistema que utilizam o arquivo.

7 - Manutenção da Integridade - Consiste em impedir que um determinado código ou chave em uma tabela não tenha correspondência em outra tabela. Ex. Um código de uma determinada disciplina na tabela “Histórico Escolar” sem a sua descrição na tabela “Disciplina”.

ANOTAÇÃO 2 Tipos de Banco de Dados

2.1 Modelo de Rede – Conceito básico.

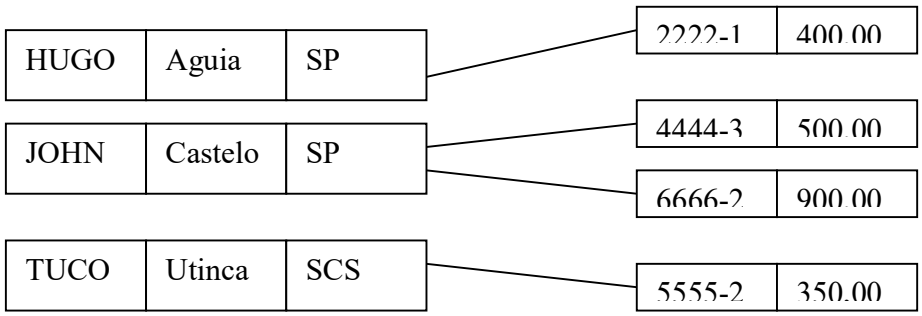
No Modelo de banco de dados em Rede os dados são representados por uma coleção de ***Registros*** e os relacionamentos entre os dados são representados por meio de ***links***.
Um banco de dados de rede é uma coleção de registros conectados uns aos outros por meio de links. Um registro é, em muitos aspectos, similar a uma entidade do modelo entidade-relacionamento (E-R). Cada Registro é uma coleção de campos (atributos), cada qual contendo somente um valor. Um link é uma associação entre exatamente dois registros. Portanto, um link pode ser visto como uma forma restrita (binária) de relacionamento no sentido do modelo E-R.
Para ilustrar, considere um banco de dados representando o relacionamento ***conta_cliente*** em um sistema bancário. Há dois tipos de registros, ***cliente*** e ***conta***. Conforme visto anteriormente, podemos definir um tipo de registro cliente usando uma notação do tipo Pascal como a seguinte:

```
Type cliente = Record
                    Nome_cliente: string;
                    Rua_cliente: string;
                    Cidade_cliente: string;
               End
```

O tipo de registro conta pode ser definido da seguinte forma:

```
Type conta  = Record
                    Numero_conta: string;
                    Saldo: integer;
               End
```

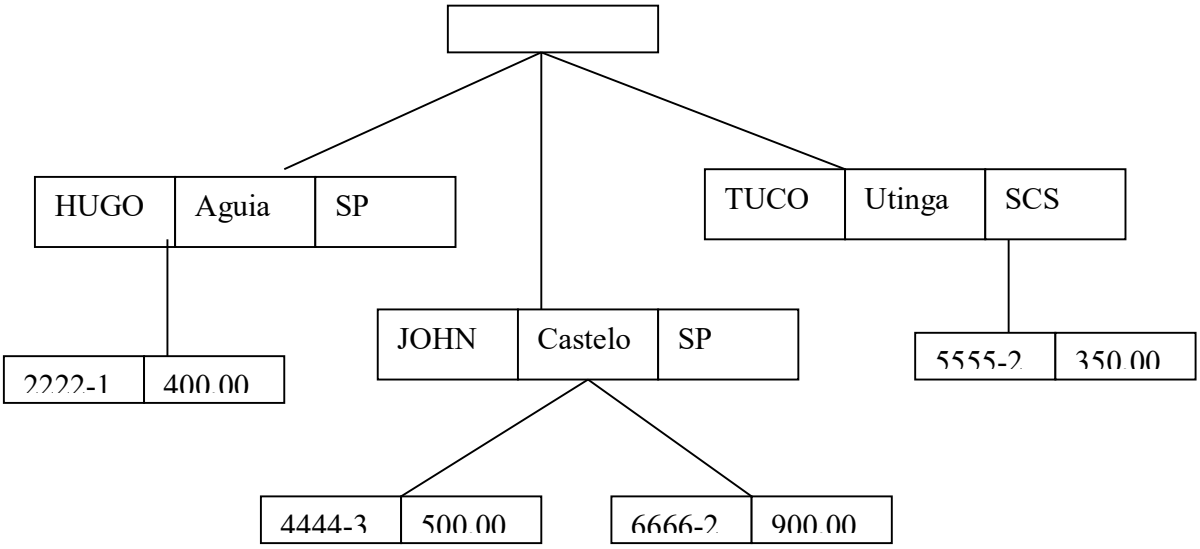
O modelo de banco de dados desenhado a seguir mostra que HUGO possui a conta “2222-1”, JOHN possui as contas “4444-3”e “6666-2” e TUCO, a conta “5555-2”.



2.2 Modelo de Hierárquico – Conceito básico.

No modelo de rede, os dados são representados por uma coleção de registros e os relacionamentos entre dados são representados por meio de links. Essa estrutura também é a estrutura do modelo hierárquico. A única diferença é que, no modelo hierárquico, os registros são organizados como coleções de árvores em vez de gráficos arbitrários.

Veja abaixo um exemplo de banco de dados



ANOTAÇÃO 3 TIPOS DE MODELOS

3.1 Modelo Conceitual

Este modelo representa as regras de Negocio de um Sistema da Informação, geralmente em forma textual com a utilização de diagramas, quando necessário.

Nesta fase do desenvolvimento de um Sistema, deve-se levantar os requisitos e regras que norteiam a necessidade de negócio de tal forma a poder implementa-lo em um sistema de banco de dados.

3.2 Modelo Lógico

Este Modelo representa a aplicação de técnicas e ferramentas de Analise e desenvolvimento de sistemas para o entendimento das relações entre as diversas entidades de um sistema da informação. Entre diversas técnicas utilizadas estão as de Normalização, que retira as anomalias que se podem gerar na implantação de um banco de dados relacional.

3.3 Modelo Físico

Este Modelo representa a aplicação de fundamentos dos SGBDs (Sistemas Gerenciadores de Banco de Dados) para a implementação de um Sistema da Informação.

O Modelo físico representa as características que o Modelo de dados exigirá do SGBD que estará inserido, além de poder detalhar requisitos de rendimento para o bom desempenho da aplicação que o utilizará. Exemplo: Tipos de dados, relações exclusivas de integridade, volume de dados, tempos de resposta esperados, etc.

Pode ser entendido pelo conjunto de desenhos físicos, juntamente com os scripts que geram o banco de dados (DDLs – Data Definition Language).

ANOTAÇÃO 4 NORMALIZAÇÃO

- Conceito introduzido por Codd (1970);
- Técnica de projeto de BD para evitar anomalias do modelo relacional;
- Foram criadas inicialmente 3 formas normais;
- Motivação principal: características de um mau projeto
  - Repetição de Informação
  - Inabilidade de representar certas informações
  - Perda de informação

- Tipos
  - 1ª Forma Normal
  - 2ª Forma Normal
  - 3ª Forma Normal
  - FNBC (Forma Normal de Boyce-Codd)
  - 4ª Forma Normal
  - 5ª Forma Normal

4.1 Problemas da falta de normalização:

Anomalias de inclusão, exclusão e alteração;

- Pedido (**Num_ pedido** , prazo, cliente, endereco, cidade, uf, insc_ est, (cod_produto, unid, qtde, desc, val_unit, total_ prod)*, total_ pedido, cod_vendedor, nome_vendedor)

– **Anomalia de inclusão** : ao ser incluído um novo cliente, o mesmo tem que estar relacionado a uma venda.
– **Anomalia de exclusão** : ao ser excluído um cliente, os dados referentes às suas compras serão perdidos.
– **Anomalia de alteração** : caso algum fabricante de produto altere o preço de um produto, será preciso percorrer toda a relação para se realizar múltiplas alterações.

4.2 1ª Forma Normal

Relações sem **atributos** multivalorados.

• Pedido (**Num_ pedido** , prazo, cliente, endereco, cidade, uf, insc_ est, **(cod_ produto, unid, qtde, desc, val_ unit, total_ prod)***, total_ pedido, cod_ vendedor, nome_ vendedor)

Um pedido pode ter muitos códigos de produtos. Portanto manter a informação de produtos na mesma entidade provocará uma repetição das informações do pedido para cada produto do mesmo pedido.
Aplicando a 1ª Forma Normal , podemos eliminar esse problema criando uma entidade exclusiva para códigos de produtos que faça um ***“relacionamento”*** com a entidade de Pedidos, ficando da seguinte forma.

•Pedido(**Num_ pedido** , prazo, cliente, endereco, cidade, uf, insc_est, total_ pedido, cod_ vendedor, nome_ vendedor)
• Item_ Pedido(**num_ pedido** , **cod_ produto** , unid, Quantidade, desc, val_unit, total_prod)

Entidade: Pedido
Num_pedido ***(Primary Key)***
Prazo
Cliente
Endereço
Cidade
Uf
Insc_est
Total_pedido
Cod_vendedor
Nome_vendedor

Entidade: Item Pedido
Num_pedido ***(Foreign Key)***
Cod_produto
Unidade
Quantidade
Descricao
Valor_Unitario
Total_prod

4.3 Dependência

Observe a dependência que existe nos atributos relacionados abaixo referente ao esquema de Pedidos e Item_Pedidos, citado acima:

num_ pedido ← prazo
cod_vendedor ← nome_vendedor

4.3.1 Dependência Funcional Total

Ex:. ***Quantidade*** depende de forma total de ***num_ pedido*** + ***cod_ produto***

4.3.2 Dependência Funcional Parcial

Ex:. ***Descrição*** depende de cod_ produto e não de num_ pedido + cod_ produto

4.3.3 Dependência Funcional Transitiva

Ex:. ***nome_vendedor*** depende de ***cod_vendedor*** que não é chave primária
mas depende desta.

4.4 2ª Forma Normal

São relações na 1ª Forma normal , e sem dependência Parcial.

• Item_ Pedido(**num_ pedido** , **cod_ produto** , unidade, quantidade, desc, val_unit, total_prod)

Aplicando a 2ª Forma Normal, podemos eliminar a dependência parcial que existe para a ***descrição*** de ***cod_produto.***

• Item_ Pedido(**num_ pedido** , **cod_ produto** , quantidade, total_ prod)
• Produto(**cod_produto,** unidade, descricao, val_unit)

Entidade: **Item_Pedido**
Num_pedido ***(Foreign Key)***
Cod_produto ***(Foreign Key)***
Quantidade
Total_produto

Entidade: **Produto**
Cod_produto ***(Primary Key)***
Unidade
Descricao
Valor_Unit

4.5 3ª Forma Normal

São Relações na 2ª Forma Normal e sem dependência transitiva

Seja a entidade Pedido:
•Pedido(**Num_ pedido** , prazo, cliente, endereco, cidade, uf, insc_ est, total_ pedido, cod_ vendedor, nome_ vendedor)

Aplicando a 3ª forma normal, podemos eliminar as dependências transitivas que existem de ***cliente*** e ***vendedor***.

• Pedido(**Num_pedido** , prazo, cod_cliente, total_ pedido, cod_vendedor)
• Cliente(**cod_cliente** , nome, endereco, cidade, uf, insc_est)
• Vendedor(**cod_vendedor** , nome_vendedor)

Entidade: **Pedido**
Num_pedido ***(Primary Key)*** Prazo Cod_cliente ***(Foreign Key)*** Total_pedido Cod_vendedor

Entidade: **Cliente**
Cod_cliente ***(Primary Key)*** nome endereco cidade uf insc_est

Entidade: **Vendedor**
Cod_Vendedor ***(Primary Key)*** Nome_vendedor

4.6 Regra Geral da Normalização

“Todo item de dados em uma relação é dependente da chave, da chave toda e de nada mais do que a chave”
[James Martin, 1991]

“Um bom modelo de dados gera relações em 3FN”

4.7 Forma Normal de Boyce- Codd (FNBC)

É uma forma mais restritiva da 3ª Forma Normal, isto é, toda relação na FNBC também esta na 3ª Forma Normal. Entretanto, uma relação na 3ª Forma Normal não necessariamente está na FNBC.

Uma relação esta na FNBC se para toda **df X → Z** (dependência funcional), X é uma super-chave.

Exemplo 1: Seja as entidades na 3ª forma normal:
Agência (**ag_nome** , ativos, cidade)
ag_nome ← ativos, cidade

Cliente (**nome_cli** , rua, cidade)
nome_cli ← rua, cidade

Seja as entidades não na FNBC abaixo:

* Depósito (ag_nome, **conta_num** , nome_cli, saldo)
conta_ num ← saldo, ag_nome
* Empréstimo (ag_nome, **emp_num** , nome_cli, quantia)
emp_num ← quantia, ag_nome

Aplicando a **FNBC**, temos:

Solução:
Info_Empréstimo(ag_ nome, **emp_num** , quantia)
Cliente_Empréstimo(**nome_cli, emp_num**)
Info_conta(ag_nome, **conta_num** , saldo)
Cliente_conta(**nome_cli, conta_num**)

No caso acima, apesar de estar na FNBC ocorreu redundância de, por exemplo, ***nome_cli.***

Exemplo 2: Seja a entidade ***AULA*** na 3ª forma normal:

AULA
ALUNO DISCIPLINA PROFESSOR

Dependência Funcional 1: Professor ← Disciplina (**Professor não** é super-chave)
Dependência Funcional 2: Professor ← Aluno, Disciplina (**Professor não** é super-chave)

Exemplo de dados para o caso acima:
Entidade: aula

ALUNO	**DISCIPLINA**	**PROFESSOR**
Carlos	Química	Ana
Carlos	Física	Antonio
Marta	Química	Ana
Marta	Português	Maria
João	Português	Manoel

Anomalia de exclusão: **Se Carlos sair da aula de Física, não teremos nenhum registro de que Antonio leciona Física**.
Portanto , vamos aplicar a FNBC que transformará a entidade AULA em duas outras (aluno_professor e professor_disciplina):

Entidade **Aluno_professor** com dados:

ALUNO	**PROFESSOR**
Carlos	Ana
Carlos	Antonio
Marta	Ana
Marta	Maria
João	Manoel

Entidade **professor_disciplina** com dados:

PROFESSOR	**DISCIPLINA**
Ana	Química
Antonio	Física
Maria	Português
Manoel	Português

Pode-se observar que não ocorre mais a anomalia de exclusão citada acima. Caso Carlos saia da aula de Física, cujo professor é Antônio, não mais perderemos a informação de que Antonio é professor de Física.

Note, entretanto, que a dependência funcional 2: Professor ← Aluno, Disciplina foi perdida com a normalização.

Considere, por exemplo, a inserção de (Marta, Manoel) na entidade **Aluno_professor**. Provocará uma **anomalia** expressando que Marta estuda português com 2 (dois) professores distintos. Isto pode estar ferindo uma regra de negócio.

CONCLUSÃO: Decomposição / Junção sem Perda.

Alguns esquemas não podem ser normalizados em FNBC e, ao mesmo tempo, preservar todas as dependências funcionais.
Solução: Redundância Parcial.

Logo faríamos a aplicação da FNBC. Porém com redundância da informação de professor na entidade **aluno_disciplina** (que poderíamos chamar simplesmente de entidade **aula**).
Veja como ficaria

Entidade: aula

ALUNO	**DISCIPLINA**	**PROFESSOR**
Carlos	Química	Ana
Carlos	Física	Antonio
Marta	Química	Ana
Marta	Português	Maria
João	Português	Manoel

Entidade **professor_disciplina** com dados:

PROFESSOR	DISCIPLINA
Ana	Química
Antonio	Física
Maria	Português
Manoel	Português

E ficamos com uma relação na FNBC, com redundância parcial e todas as dependências funcionais preservadas.

4.8 4ª Forma Normal

Existe **Dependência** Multivalorada.
Uma relação esta em 4ª forma normal , se e somente se, estiver também na FNBC.

Exemplo: Livros(**num_chamada, autor, assunto**)
Aplicando 4ª Forma normal temos:

Livros1(**num_chamada** , autor)
Livros2(**num_chamada** , assunto)

num_ chamada	autor	assunto
1	autor1	assunto1
1	autor1	assunto2
1	autor2	assunto1
1	autor2	assunto2
2	Autor1	assunto3
2	Autor1	assunto4

No exemplo acima o atributo ***assunto*** tem dependência funcional multivalorada do atributo ***autor***. A solução apresentada após a 4ª forma normal foi a separação em 2 entidades distintas para se evitar a repetição do atributo ***autor***.

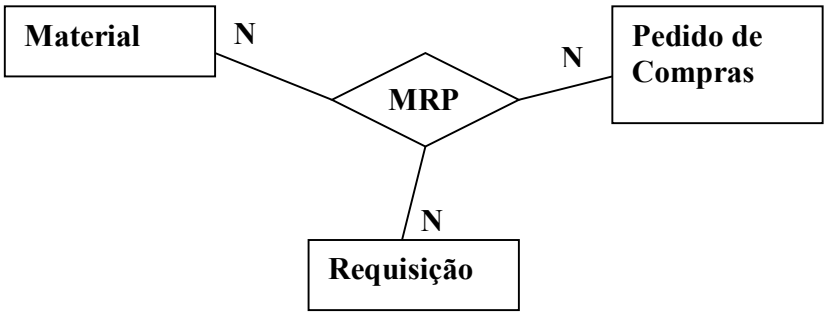
Entenda **num_chamada** como uma coleção de livros. Por exemplo, a Barsa.

4.9 5ª Forma Normal

Ocorre Dependência de junção - relacionamentos n- ários
Uma relação está em 5ª Forma Normal quando o seu conteúdo não puder ser reconstruído a partir de esquemas menores.
Exemplo: MRP(material, requisição, pedido)
– Desmembramento:

- Mat_pedido(material, pedido)
- Ped_requisição(pedido, requisição)
- Mat_requisição(material, requisição)

– Se na junção houver preservação de informações, então a solução é manter o desmembramento



Entidade: MRP

Material	Pedido	Requisição
M1	P1	R1
M1	P2	R2
M2	P1	R2
M1	P1	R2

Entidade: MRP aplicada a 5ª Forma Normal:

Material	Pedido
M1	P1
M1	P2
M2	P1

Pedido	Requisição
P1	R1
P2	R2
P1	R2

Material	Requisição
M1	R1
M1	R2
M2	R2

Exercícios de Normalização:

1 – (Modelo Conceitual) - Defina um modelo de dados para uma Empresa de Produção de automóveis. Reunir-se em grupos de 3 a 4 pessoas para discutirem e relacionarem as Entidades, atributos, e relacionamentos que expressem as regras de negócios discutidas. Entregar folha do grupo.

Passos para Iniciação de criação de um modelo de dados:

1. Identifique o número máximo de entidades que acredite existir;
2. Defina o relacionamento entre as distintas entidades;
3. Determine todos os atributos que acredite ser viável para cada entidade.

PS: Não considere Normalização neste momento.

ANOTAÇÃO 5 REGRAS DE INTEGRIDADE

“As regras de integridade fornecem a garantia de que mudanças feitas no banco de dados por usuários autorizados não resultem em perda da consistência de dados” [Abraham Silberschatz - 1991].

5.1 Restrições de Domínio

Um conjunto de valores válidos pode ser associado a um atributo (coluna) de uma tabela. A este conjunto chamamos de domínio. Quando há um domínio predefinido para um campo dizemos que este campo tem ***restrição de domínio***.

Exemplo: Seja o atributo ***Sexo***. Podemos definir um conjunto domínio para este atributo que seja “M” ou “F” , de Masculino e Feminino respectivamente , ou “1” e “2”, caso a representação numérica seja melhor. Neste exemplo criamos uma restrição de domínio para o Atributo Sexo, onde qualquer outro valor colocado no campo resultaria em um Erro disparado pelo SGBD.

Um domínio também pode ser especificado de forma condicional durante a criação da Tabela através da cláusula **CHECK(<condição de restrição de dominio>)** ..
Exemplo:

> **Atributo >= 4000**. Portanto, todo o valor que for inserido para o atributo deve ser maior ou igual a 4000, caso contrario o SGBD dispararia uma mensagem de erro.
>
> **CHECK (atributo >= 4000)**

5.2 Integridade Referencial

Freqüentemente, desejamos garantir que um valor que aparece em uma relação para um dado conjunto de atributos também apareça para um certo conjunto de atributos de outra relação. Essa condição é chamada ***integridade referencial***.

Na prática, Integridade referencial é o nome que damos à garantia de consistência de dados, devido às ***dependências funcionais***, que as Chaves Primarias e Chaves Estrangeiras estabelecem no banco de dados relacional às suas relações.

O SGBD (Sistema Gerenciador de Banco de Dados), através de comandos SQL, pode estabelecer estas chaves e garantir a consistência dos dados estabelecidas entre elas de forma automática. A esta característica chamamos de ***Integridade Referencial declarativa***, por foi estabelecida por uma declaração SQL.

5.2.1 Chaves Primárias (Primary keys)

Garantem a unicidade das informações em suas relações básicas, e são exportadas para tabelas a que tem chave estrangeira relacionada.
exemplos:
Um FUNCIONÄRIO só pode ser cadastrado uma única vez em seu cadastro básico.
Um ALUNO possui somente uma linha na tabela de ALUNOS.

Uma chave primária é o identificador único, capaz de definir a relação na Tabela.

As CHAVES PRIMÁRIAS são componentes imprescindíveis para garantir a *INTEGRIDADE REFERENCIAL* das tabelas.

Quando uma tabela tem mais de um atributo que, sozinho, possa garantir a unicidade do registro, dizemos que a Tabela tem mais de uma ***chave candidata***.
Chave Candidata é aquela que possui todas as características para se tornar a chave primária da Tabela. Obviamente quando uma tabela somente tem uma chave candidata esta mesma será a chave primária.

5.2.2 Chaves estrangeiras ou externas (Foreign Keys)

Completam a *INTEGRIDADE REFERENCIAL* das relações, ou seja, sem a declaração de uma Chave externa, a Integridade referencial não esta implementada.

Uma relação R2 se referencia com uma relação R1 por uma *chave externa*, que deve ter seu valor igual ao valor da *chave primária* da segunda (R1).

Para garantir as duas regras citadas, o SQL dispõe de declarações para definir as chaves *PRIMÁRIAS* e *EXTERNAS.*

5.3 Gatilhos (Triggers)

Gatilhos são procedimentos executados a partir de um evento de Update, Insert ou Update. Um Gatilho é um comando que é executado pelo sistema automaticamente, em conseqüência de uma modificação no banco de dados. Duas exigências devem ser satisfeitas para a projeção de um mecanismo de gatilho:

1. Especificar as condições sob as quais o gatilho deve ser executado;
2. Especificar as ações que serão tomadas quando um gatilho for disparado.

Variavelmente, ou até mesmo por conta de características de alguns SGBD, os gatilhos são utilizados para garantir a integridade referencial de uma determinada relação. O efeito e custos de se utilizar a integridade referencial declarativa ou por meio de gatilhos são semelhantes. No entanto os gatilhos são mais flexíveis porque podem executar ações

distintas antes ou depois de garantir a integridade, como por exemplo, enviar email, atualizar tabelas de Controle, etc.

ANOTAÇÃO 6 INTRODUÇÃO SQL

A linguagem SQL (Structured Query Language) representa um conjunto de comandos responsáveis pela definição das estruturas (tais como: tabelas, índices e relações de integridades), comandos de controle e atualização dos dados em um S.G.B.D.(Sistema Gerenciador de Banco de Dados). A linguagem SQL veio para padronizar e simplificar o uso dos Banco de dados Relacionais por volta do inicio da década de 70.

Os comandos existentes nesta linguagem que serão abordados neste curso estão subdivididos em três grupos:

- Comandos DDL (Data Definition Language) - Conjunto de comandos responsáveis pela criação, alteração e exclusão da estrutura das tabelas e índices de um sistema.
- Comandos DML (Data Manipulation Language) - Conjunto de comandos responsáveis pela consulta e atualização dos dados armazenados em um banco de dados.
- Comandos DCL (Data Control Language) – Conjunto de comandos responsáveis pelo controle de permissões de acesso às tabelas e demais estruturas do banco de dados.

6.1 Comandos DDL

6.1.1 Create Table

Objetivo:

Criar a estrutura de uma tabela definindo as colunas e as chaves primárias e estrangeiras existentes.

Sintaxe:

CREATE TABLE <nome-tabela>

(<nome-coluna> <tipo-do-dado> [NOT NULL],

[NOT NULL WITH DEFAULT])

onde:

a) nome-tabela - Representa o nome da tabela que será criada.

b) nome-coluna - Representa o nome da coluna que será criada. A definição das colunas de uma tabela é feita relacionando-as uma após a outra.

c) tipo-do-dado - Cláusula que define o tipo e tamanho do dado definido para o campo da tabela. Os tipos de dados mais comuns serão definidos adiante.

d) NOT NULL - Exige o preenchimento do campo, ou seja, no momento da inclusão é obrigatório que possua um conteúdo.

e) NOT NULL WITH DEFAULT - Preenche o campo com valores pré-definidos, de acordo com o tipo do campo, caso não seja especificado o seu conteúdo no momento da inclusão do registro. Os valores pré-definidos são:

e.1) Campos numéricos - Valor zero.

e.2) Campos alfanuméricos - Caracter branco.

e.3) Campo formato Date - Data corrente.

e.4) Campo formato Time - Horário no momento da operação.

Tipos de dados mais comuns:

1) Numéricos:

- Smallint - Armazena valores numéricos, em dois bytes binários, compreendidos entre o intervalo -32768 a +32767.

- Integer - Armazena valores numéricos, em quatro bytes binários, compreendidos entre o intervalo -2147483648 a +2147483647.

- Decimal(n,m) - Armazena valores numéricos com no máximo 15 dígitos. Onde (n) é a quantidade de dígitos da parte inteira somada à quantidade de dígitos decimais; e (m) e a quantidade de dígitos decimais.

2) Alfanuméricos:

- **Varchar(n)** - Definir um campo alfanumérico variável de até n caracteres, onde n deve ser menor ou igual a 254 caracteres (Alguns sistemas gerenciadores de banco de dados suportam um valor maior).

- **Char(n)** - Definir um campo alfanumérico fixo de n caracteres, onde n deve ser menor ou igual a 254 caracteres.

- **Long Varchar** - Definir um campo alfanumérico de comprimento maior que 254 caracteres. Os SGBDs atuais tem definições de tipos de dados que suportam valores acima de 254 caracteres. Exemplo: Oracle: varchar2 , SQLServer: text

3) **Date -** Definir um campo armazenará datas (dia, mês e ano).

4) **Time** - Definir um campo que irá armazenar de horário (hora, minuto, segundos, milisegundos).

5) **DateTime** – Definir um campo que armazenará Data e hora simultaneamente.

6) **BLOB** (Binary Large OBject) Binários – Definir um campo que irá armazenar um dado binário. (Imagens, filmes, som , etc.)

Existem diversos outros tipos de dados que variam seu nome dependendo do SGBD e que, alguns deles, em teoria pode armazenar até 2 Gbytes de informação num único campo.

6.1.2 Alter Table

Objetivo:

Alterar a estrutura de uma tabela(arquivo) acrescentando, alterando, retirando e alterando nomes, formatos das colunas e a integridade referencial definidas em uma determinada tabela.

Sintaxe:

ALTER TABLE <nome-tabela>

DROP <nome-coluna>

ADD <nome-coluna> <tipo-do-dado> [NOT NULL]
[NOT NULL WITH DEFAULT]

RENAME <nome-coluna> <novo-nome-coluna>

RENAME TABLE <novo-nome-tabela>

MODIFY <nome-coluna> <tipo-do-dado> [NULL]
[NOT NULL]
[NOT NULL WITH DEFAULT]

ADD PRIMARY KEY <nome-coluna>

DROP PRIMARY KEY <nome-coluna>

ADD FOREIGN KEY (nome-coluna-chave-estrangeira) **REFERENCES**
(nome-tabela-pai) **ON DELETE** [RESTRICT]
[CASCADE]
[SET NULL]

DROP FOREIGN KEY (nome-coluna-chave-estrangeira) **REFERENCES**
(nome-tabela-pai)

onde:

a) nome-tabela - Representa o nome da tabela que será atualizada.

b) nome-coluna - Representa o nome da coluna que será criada.

c) tipo-do-dado - Cláusula que define o tipo e tamanho dos campos definidos para a tabela.

d) DROP <nome-coluna> - Realiza a retirada da coluna especificada na estrutura da tabela.

e) ADD <nome-coluna> <tipo-do-dado> - Realiza a inclusão da coluna especificada na estrutura da tabela. Na coluna correspondente a este campo nos registros já existentes será preenchido o valor NULL (Nulo). As definições NOT NULL e NOT NULL WITH DEFAULT são semelhantes à do comando CREATE TABLE.

f) RENAME <nome-coluna> <novo-nome-coluna> - Realiza a troca do nome da coluna especificada.

g) RENAME TABLE <novo-nome-tabela> - Realiza a troca do nome da tabela especificada.

h) MODIFY <nome-coluna> <tipo-do-dado> - Permite a alteração na característica da coluna especificada.

i) PRIMARY KEY (nome-coluna-chave) - Definir para o banco de dados a coluna que será a chave primária da tabela. Caso ela tenha mais de um coluna como chave, elas deverão ser relacionadas entre os parênteses.

j) FOREIGN KEY (nome-coluna-chave-estrangeira) REFERENCES (nome-tabela-pai) - Definir para o banco de dados as colunas que são chaves estrangeiras, ou seja, os campos que são chaves primárias de outras tabelas. Na opção REFERENCES deve ser especificado a tabela na qual a coluna é a chave primária.

k) ON DELETE - Esta opção especifica os procedimentos que devem ser feitos pelo SGBD quando houver uma exclusão de um registro na tabela pai quando existe um registro correspondente nas tabelas filhas. As opções disponíveis são:

k.1) RESTRICT - Opção default. Esta opção não permite a exclusão na tabela pai de um registro cuja chave primária exista em alguma tabela filha.

k.2) CASCADE - Esta opção realiza a exclusão em todas as tabelas filhas que possua o valor da chave que será excluída na tabela pai .

k.3) SET NULL - Esta opção atribui o valor NULO nas colunas das tabelas filhas que contenha o valor da chave que será excluída na tabela pai.

Opções:

Além das existentes na opção ADD (NOT NULL e NOT NULL WITH DEFAULT), existe a opção NULL que altera a característica do campo passando a permitir o preenchimento com o valor Nulo (na maioria dos SGBD essa é a opção “default”).

i) ADD PRIMARY KEY <nome-coluna> - Esta opção é utilizada para adicionar uma chave primária da tabela.

j) DROP PRIMARY KEY <nome-coluna> - Esta opção é utilizada para excluir chave primária da tabela.

k) ADD FOREIGN KEY <nome-coluna> - Esta opção é utilizada para adicionar um novo campo que é uma chave estrangeira.

l) DROP FOREIGN KEY <nome-coluna> - Esta opção é utilizada para excluir uma chave estrangeira da estrutura da tabela.

6.1.3 Drop Table

Objetivo:

Excluir a estrutura e os dados existentes em uma tabela. Após a execução deste comando estarão excluídos todos dados, estrutura e índices de acessos que estejam a ela associados.

Sintaxe:

DROP TABLE <nome-tabela>

onde:

a) nome-tabela - Representa o nome da tabela que será excluida.

6.1.4 Truncate Table

Objetivo:

Excluir todos os registros de uma tabela. É similar ao comando DELETE sem clausula Where, porém com a diferença importante de que não é possível fazer ROLLBACK da operação;

Sintaxe:

TRUNCATE TABLE <nome-tabela>

onde:

a) nome-tabela - Representa o nome da tabela para excluir os dados.

6.1.5 Create Index

Objetivo:

Criar uma estrutura de índice de acesso para uma determinada coluna em uma tabela. Um índice de acesso permite um acesso mais rápido aos dados em uma operação de seleção. Os índices podem ser criados a partir de um ou mais campos de uma tabela.

Sintaxe:

CREATE [UNIQUE] INDEX <nome-índice>

ON <nome-tabela> (<nome-coluna> [ASC], [<nome-coluna> [ASC]])
[DESC] [DESC]

onde:

a) nome-índice - Representa o nome da estrutura de índice que será criada.

b) nome-tabela - Representa o nome da tabela que contem a coluna na qual será criada o índice de acesso.

c) nome-coluna - Representa o nome da coluna que será criada.

d) Opção ASC/DESC - Representa a criação do índice ordenada crescentemente (ASC) ou decrescentemente (DESC).

6.1.6 Drop Index

Objetivo:

Excluir uma estrutura de índice de acesso para uma determinada coluna em uma tabela.

Sintaxe:

DROP INDEX <nome-índice>

onde:

a) nome-índice - Representa o nome da estrutura de índice que será excluído.

6.2 Comandos DML

6.2.1 Insert

Objetivo:

Incluir um novo registro em uma tabela do Banco de Dados.

Sintaxe:

INSERT INTO <nome-tabela> [**(**<nome-coluna>, [<nome-coluna>]**)**]

VALUES (<relação dos valores a serem incluídos>)

onde:

a) nome-tabela - Representa o nome da tabela onde será incluído o registro.

b) nome-coluna - Representa o nome da(s) coluna(s) terão conteúdo no momento da operação de inclusão.

Obs.: Este comando pode ser executado de duas maneiras:

1) Quando todos os campos da tabela terão conteúdo não é necessário especificar as colunas, entretanto a relação dos valores a serem incluídos deve obedecer a mesma seqüência da definição da tabela. Neste caso, os campos que não terão conteúdo devem ser especificados com o valor NULL.

2) Quando apenas parte dos campos da tabela serão incluídos, devem ser especificadas todas as colunas que terão conteúdo e os valores relacionados deverão obedecer esta seqüência.

6.2.2 Update

Objetivo:

Atualiza os dados de um ou mais registros de uma tabela do Banco de Dados.

Sintaxe:

UPDATE <nome-tabela>

SET <nome-coluna> = <novo conteúdo para o campo>

[<nome-coluna> = <novo conteúdo para o campo>]

WHERE <condição>

onde:

a) nome-tabela - Representa o nome da tabela cujo conteúdo será alterado.

b) nome-coluna - Representa o nome da(s) coluna(s) terão seus conteúdos alterados com o novo valor especificado.

c) condição - Representa a condição para a seleção dos registros que serão atualizados. Esta condição pode resultar em um ou mais registros. ATENÇÃO: Se a cláusula WHERE não for especificada o comando fará alteração em TODOS os registros da tabela.

6.2.3 Delete

Objetivo:

Excluir um ou mais registros em uma tabela do Banco de Dados.

Sintaxe:

DELETE FROM <nome-tabela>

WHERE <condição>

onde:

a) nome-tabela - Representa o nome da tabela cujos registros serão excluídos.

b) condição - Representa a condição para a exclusão dos registros. Esta condição pode resultar na exclusão de um ou mais. ATENÇÃO: Se a cláusula WHERE não for especificada o comando excluirá TODOS os registros da tabela.

6.2.4 Select

Objetivo:

Selecionar um conjunto de registros em uma ou mais tabelas/visões que atenda a uma determinada condição definida pelo comando.

Sintaxe:

SELECT [ALL / DISTINCT] <nome das colunas / * >

FROM <nome-tabela/visão> [, <nome-tabela/visão>]

WHERE <condição>

GROUP BY <nome-coluna>

HAVING <condição>

ORDER BY <nome-campo> ASC
DESC

onde:

a) nome-tabela/visão - Representa o nome da(s) tabela(s) e/ou visões que contêm as colunas que serão selecionadas ou que serão utilizadas para a execução da consulta.

b) condição - Representa a condição para a seleção dos registros. Este seleção poderá resultar em um ou vários registros.

c) nome-coluna - Representa a(s) coluna(s) cujos resultados são grupados para atender à consulta.

d) ALL - Opção default. Mostra todos os registros obtidos na seleção.

e) DISTINCT - Opção que mostra os registros obtidos na seleção eliminando as duplicidades.

f) WHERE - Especifica o critério de seleção dos registros nas tabelas especificadas.

g) GROUP BY - Especifica o(s) campo(s) que serão grupados para atender a consulta.

h) HAVING - Especifica uma condição para seleção de um grupo de dados. Esta opção só é utilizada combinada com a opção GROUP BY.

i) ORDER BY - Esta opção quando utilizada apresenta o resultado da consulta ordenado de forma crescente ou decrescente pelos campos definidos.

Algumas funções **AGREGADAS** utilizadas no comando Select.

a) **COUNT**(*)

(DISTINCT <nome-campo>)

Objetivo:

Retorna a quantidade de registros existentes no campo especificado. Quando a opção * é utilizada o resultado é a quantidade de registros existentes. Quando é referenciado o nome de um campo retorna a quantidade de valores existentes para a coluna.

b) **SUM** (ALL <nome-campo>)

DISTINCT

Objetivo:

Retorna a soma dos valores existentes no campo especificado. Quando a opção DISTINCT é utilizada são considerados apenas os registros cujo resultado sejam distintos.

c) **AVG** (ALL <nome-campo>)

DISTINCT

Objetivo:

Retorna a média dos valores existentes no campo especificado. Quando a opção DISTINCT é utilizada considerados apenas os registros cujo resultado sejam distintos.

d) **MAX** (ALL <nome-campo>)

DISTINCT

Objetivo:

Retorna o maior valor existente no campo especificado. Quando a opção DISTINCT é utilizada são considerados apenas os registros cujo resultado sejam distintos.

e) **MIN** (ALL <nome-campo>)

DISTINCT

Objetivo:

Retorna o menor valor existente no campo especificado. Quando a opção DISTINCT é utilizada são considerados apenas os registros cujo resultado sejam distintos.

6.2.4.1 INNER JOIN

A cláusula INNER JOIN é usada quando se quer recuperar dados em mais de uma tabela através da igualdade de suas foreign keys.

Por exemplo, pense que você quer criar um programa que sirva como agenda telefônica. Você cadastra as pessoas e os contatos telefônicos, como uma pessoa poderá ter nenhum até N números de telefones, você separa em duas entidades como neste modelo:

Olhando rapidamente você percebe que a entidade TELEFONE tem o atributo NUM_PESSOA que é o número de identificação do registro na entidade PESSOA.

Vamos supor que as tabelas estejam preenchidas com os seguintes registros:

```
+----------------------------+  +----------------------------+
|           PESSOA           |  |          CONTATO           |
+------------+---------------+  +------------+---------------+
| NUM_PESSOA | NOME_PESSOA   |  | NUM_PESSOA | NUM_TELEFONE |
| 1          | José          |  | 1          | 6589-3666    |
| 2          | Arnaldo       |  | 3          | 9888-6699    |
| 3          | Maria         |  | 3          | 8956-6666    |
| 4          | Uóchinto      |  | 2          | 2888-9877    |
+------------+---------------+  +------------+--------------+
```

Para recuperar os telefones de uma pessoa você deve usar INNER JOIN, implicitamente ou explicitamente. Vou explicar os dois casos.

O método explícito (padrão ANSI 99) é usar a cláusula explicitamente (Já ta aprendendo ☺!), que é formada por INNER JOIN <tabela filha> ON <atributos de identificação>, no nosso exemplo:

```
SELECT nome_completo, num_telefone FROM pessoa INNER JOIN
telefone ON pessoa.num_pessoa = telefone.num_pessoa
```

É o mesmo que dizer: “Selecione o nome e os números dos telefones da pessoa onde o número da pessoa em **TELEFONE** seja igual ao número da pessoa em **PESSOA**”.

Na forma implícita (modelo ANSI 92) você não usa a cláusula INNER JOIN, o tratamento de igualdade é feito na cláusula WHERE. Usando o mesmo exemplo:

SELECT nome_completo, num_telefone **FROM** pessoa, telefone **WHERE** **pessoa.num_pessoa = telefone.num_pessoa;**

Em ambos os casos o resultado será:

```
+--------------------------+
|         CONTATO          |
+------------+-------------+
| NOM_PESSOA | NUM_TELEFONE |
| José       | 6589-3666    |
| Maria      | 9888-6699    |
| Maria      | 8956-6666    |
| Arnaldo    | 2888-9877    |
+------------+-------------+
```

“Muito simples! Hummmm mas péra ai… Onde está o Uóchinto (que nominho, hein?!).”

Lembra que dissemos ao banco “Seleciona o nome e os números dos telefones da pessoa onde o número da pessoa em TELEFONE seja igual ao número da pessoa em PESSOA”? Então! O Uóchinto não esta na tabela TELEFONE, por isso não foi devolvido no resultado.

Quando você precisar retornar os dados, mesmo não estando na tabela filha, usamos o OUTER JOIN.

6.2.4.2 OUTER JOIN

A cláusula OUTER JOIN tem duas variantes, LEFT OUTER JOIN e RIGHT OUTER JOIN, sendo que estas cláusulas devem ser declaradas sempre de maneira explícita. Nas primeiras versões de SGBDs facilitavam a utilização do OUTER JOIN, de acordo ao

modelo ANSI 92, não exigindo declarar esta sintaxe, como é o caso do Oracle que utiliza o símbolo (+) na identificação fraca, ou mesmo o Microsoft SQLServer que utiliza (*) mas NÃO É do ANSI 99. Ou seja, existe deficiência na utilização desses “mnemônicos” de forma que para OUTER JOIN de varias tabelas com junção de varias colunas não funcionariam corretamente, e não será detalhada nesta anotação a razão, por não ter importância didática. A forma correta é utilizar o texto OUTER JOIN (LEFT OU RIGHT), explicitamente.

Para que apareçam todas as pessoas, inclusive as que não têm número de telefone cadastrados usamos LEFT OUTER JOIN ou RIGHT OUTER JOIN.

A sintaxe é parecida com a INNER JOIN, mudando apenas o nome da cláusula:
LEFT[RIGHT] OUTER JOIN <tabela filha> ON <atributos de identificação>;

Veja a seguir:
SELECT nome_completo, num_telefone FROM **pessoa LEFT OUTER JOIN telefone** ON pessoa.num_pessoa = telefone.num_pessoa

Ou

SELECT nome_completo, num_telefone FROM **telefone RIGHT OUTER JOIN pessoa** ON pessoa.num_pessoa = telefone.num_pessoa

Em ambos os casos o resultado será:

```
+------------------------------+
 |          CONTATO             |
 +---------------+--------------+
 | NOM_PESSOA    | NUM_TELEFONE |
 | José          | 6589-3666    |
 | Maria         | 9888-6699    |
 | Maria         | 8956-6666    |
 | Arnaldo       | 2888-9877    |
 | Uóchinto      |              |
 +---------------+--------------+
```

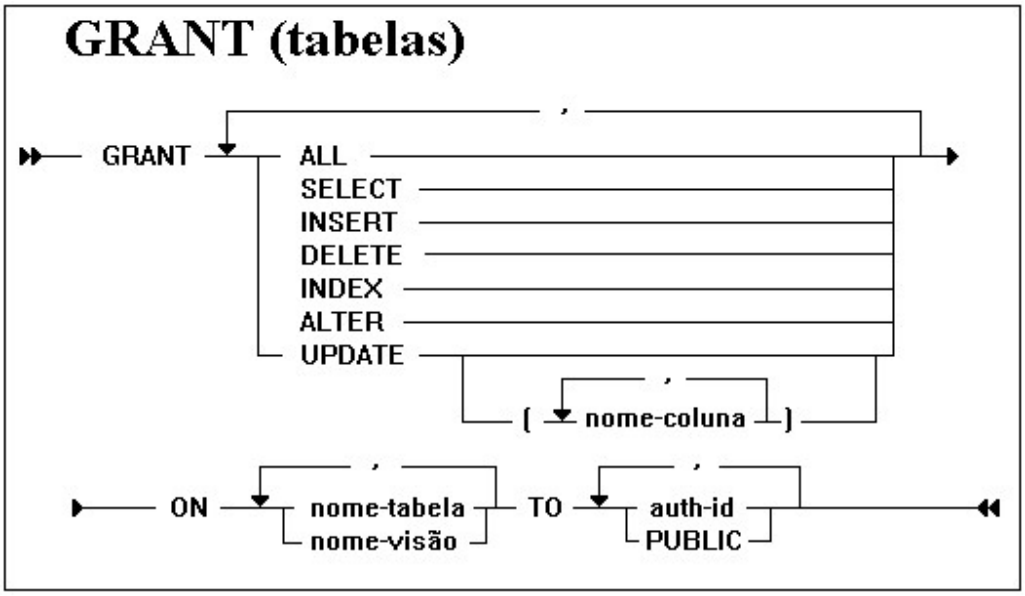
Como você pôde perceber, a única diferença entre o LEFT OUTER JOIN e o RIGHT OUTER JOIN é só a indicação de qual é o lado forte do relacionamento, ou seja, em qual lado da cláusula está a tabela que pode não ter valores na tabela filha. No nosso caso, ambos apontando para a tabela PESSOA.

INNER JOIN é muito usado para consulta em banco de dados. Já as cláusulas OUTER JOIN também são usadas, mas não com menor freqüência.

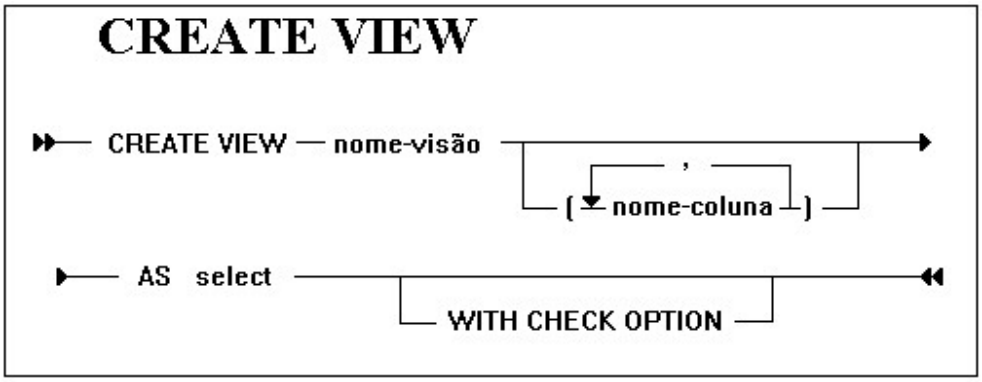
ANOTAÇÃO 7 Segurança

A segurança no acesso aos dados no SQL pode ser controlada desde o nível mais alto, em nível de DATABASE, até ao acesso à menor informação armazenada em uma TABELA, a COLUNA.

Veja abaixo a sintaxe do comando para conceder privilégios em tabelas



Uma outra maneira de restringir o acesso às informações é através de VISÕES, como mostra a sintaxe abaixo. O parâmetro opcional WITH CHECK OPTION é utilizado para garantir a integridade dos dados numa eventual tentativa de alteração dos dados, através da Visão.
Por exemplo se um comando “Insert” na Visão, provocar uma inconsistência de dados de tal forma a que o registro inserido não satisfaça a condição do “Select” que gera a própria visão, utilizando o parâmetro WITH CHECK OPTION, o banco de dados não permitirá tal comando “Insert”


CREATE VIEW
CREATE VIEW
nome-visão
,
(
nome-coluna
)
AS
select
WITH CHECK OPTION

ANOTAÇÃO 8 Transações

“Normalmente, considera-se que um conjunto de várias operações no banco de dado sé uma única unidade do ponto de vista do usuário. Por exemplo, a transferência de fundos de uma conta corrente para uma poupança é uma operação única sob o ponto de vista do cliente; dentro do sistema de banco de dados, porém, ela envolve várias operações. Evidentemente, é essencial a conclusão de todo o conjunto de operações, ou que, no caso de uma falha, nenhuma delas ocorra. Seria inaceitável o débito na conta sem o crédito na poupança.” [“Sistema de Banco de dados”, Silberschatz, Abraham].

8.1 Conceito de Transação

Uma ***transação*** é uma ***unidade*** de execução de programa que acessa e, possivelmente, atualiza vários itens de dados. Uma transação, geralmente, é o resultado da execução de um programa de usuário escrito em uma linguagem de manipulação de dados de alto nível ou em uma linguagem de programação (por exemplo, Transact-SQL , PL/SQL, Visual Basic, C ou Pascal), e é delimitada por declarações (ou chamadas de função) da **forma begin transaction** e **end transaction**. A transação consiste em todas as operações ali executadas, entre o começo e o fim da transação.

8.2 Propriedades da Transação

Para assegurar a integridade dos dados, exigimos que o sistema de banco de dados mantenha as seguintes propriedades das transações:

- **Atomicidade**. Ou todas as operações da transação são refletidas corretamente no banco de dados ou nenhuma o será.
- **Consistência**. A execução de uma transação isolada (ou seja, sem a execução concorrente de outra transação) preserva a consistência do banco de dados.
- **Isolamento**. Embora diversas transações possam ser executadas de forma concorrente, o sistema garante que, para todo par de Transações **T*i*** e **T*j***, Ti tem a sensação de que **T*j*** terminou sua execução antes de **T*i*** começar, ou que **T*j*** começou sua execução após **T*i*** terminar. Assim, cada transação não toma conhecimento de outras transações concorrentes no sistema.
- **Durabilidade**. Depois da transação completar-se com sucesso, as mudanças que ela faz no banco de dados persistem, até mesmo se houver falhas no sistema (persistência).

Estas propriedades são freqüentemente chamadas de **ACID**.

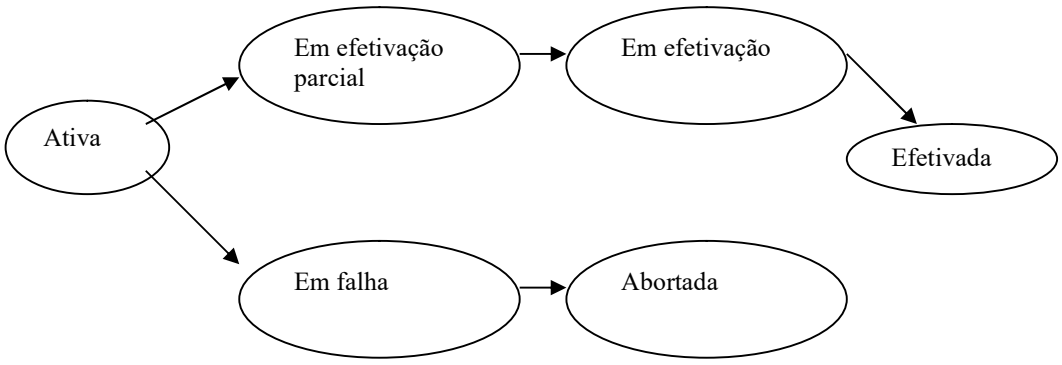
8.3 Estado da Transação

Na ausência de falhas, todas as transações completam-se com sucesso. Entretanto, como observamos anteriormente, nem sempre uma transação pode completar-se com sucesso. Nesse caso, a transação é ***abortada***. Se assegurarmos a propriedade de atomicidade, uma transação abortada não deve ter efeito algum sobre o estado do banco de dados. Assim, quaisquer atualizações que a transação abortada tiver feito no banco de dados devem ser desfeitas. Uma vez que as mudanças causadas por uma transação sejam desfeitas, dizemos que a transação foi **desfeita** (***rolled back - retornada***). Gerenciar transações abortadas é responsabilidade do esquema de recuperação.

Uma transação completada com sucesso é chamada ***efetivada*** (***committed***). Uma transação que foi efetivada e que realizou atualizações transforma o banco de dados em um novo estado consistente que deve persistir até mesmo se houver uma falha de sistema.
Uma vez que uma transação chegue à efetivação (commit), não podemos desfazer seus efeitos abortando-a. O único modo de desfazer os efeitos de uma transação efetivada é executar uma transação de compensação, porém nem sempre isso é possível. Logo, a responsabilidade pela criação e execução de uma transação de compensação é deixada a cargo do usuário, não sendo tratada pelo sistema de banco de dados.

Podemos estabelecer os seguintes estados de transação:

- **Ativa**, ou estado inicial; a transação permanece neste estado enquanto estiver executando.
- **Em efetivação parcial**, após a execução da última declaração.
- **Em falha**, após a descoberta de que a execução normal já não pode se realizar.
- **Abortada**, depois que a transação foi desfeita, através do comando **ROLLBACK**, e o banco de dados foi restabelecido ao estado anterior do início da execução da transação.
- **Em efetivação**, após a conclusão com sucesso.
- **Efetivada,** depois que a transação foi concluída com sucesso e executada o **COMMIT**.



ANOTAÇÃO 9 Triggers e Stored Procedures

9.1 Stored Procedures

É um conjunto de declarações SQL, utilizando-se do conceito de linguagem procedural, para executar tarefas de acesso ao banco de dados .

Algumas características:

- A execução ocorre sob o controle do servidor de banco de dados ;
- Todos os comandos (declarações SQL) estão, no momento da execução, compilados e podem ter seu plano de execução previamente estabelecido.
- Tem os principais recursos de programação :
  - Comandos de decisão (if , else, else if);
  - Comandos de laço (While , loop, for);
  - Operadores de atribuição/comparação (= , := ,<,>, <=,etc);
  - Operadores booleanos (and, or);
  - etc.

Utiliza-se Stored Procedure ,também , para controlar o acesso ao dado, limitando-os. Ou seja, os usuários apenas podem executar Stored Procedure criadas pelo DBA ou pela equipe responsável pelo acesso ao DB ou ao Sistema em questão.

As “linguagens procedurais” têm diferentes sintaxes, dependendo do SGBD (Sistema Gerenciado de Banco de dados envolvido) . Por exemplo, para o Microsoft SQLServer é utilizada uma linguagem chamada **“Transact-SQL”** , no caso Oracle utiliza-se uma chamada **“PL/SQL”** (Procedural Language SQL).

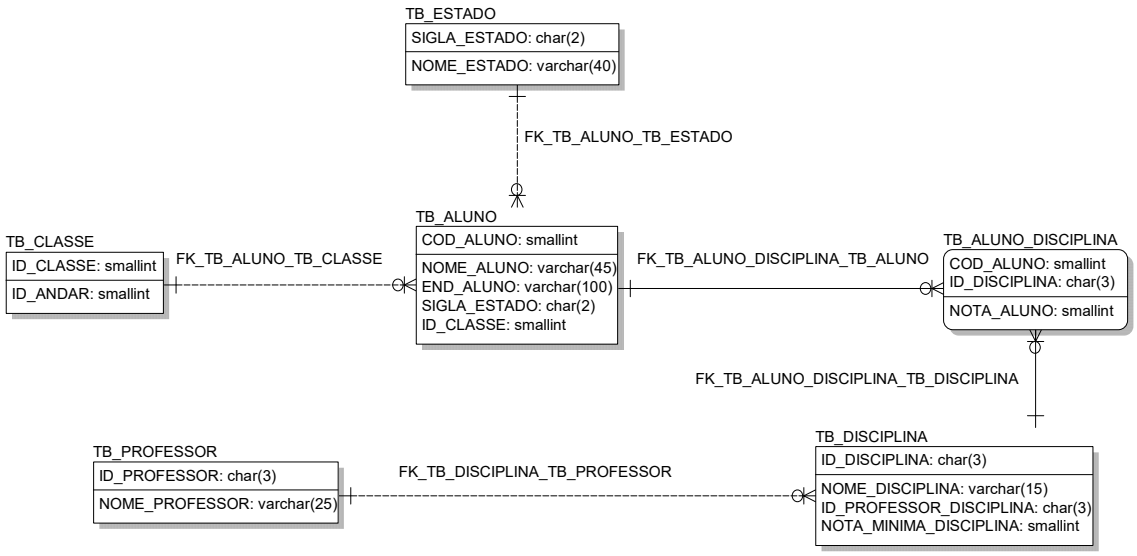
No entanto podemos dizer que a principal vantagem em usar Stored Procedure é o fato de a mesma já estar compilada no Banco de dados, o que reduz o tempo de execução, considerando que a compilação pode ocupar até 75% do tempo de execução.

9.2 Triggers

Triggers são formas especiais de Stored Procedures ,usadas para garantir integridade de dados, especialmente integridade referencial.
Triggers nunca são executadas diretamente, e sim quando há uma modificação da tabela a que pertence.
As triggers podem ser disparadas automaticamente quando efetuamos comandos “Delete”, “Insert”, ou “Update”, numa determinada tabela cujo a Trigger seja referenciada.
Uma Trigger pertence a uma única tabela. No entanto, uma tabela pode ter várias Triggers.

ANOTAÇÃO 10 Projeto Modelo de Dados CURSO

10.1 Desenho de Modelo de Dados Físico



10.2 Descrição do Conteúdo das tabelas que serão utilizadas

Após desenhar o modelo de dados físico e Criar todas as tabelas, Primary keys e Foreign keys, inserir dados nas mesmas de acordo com as definições de conteúdo abaixo, utilizando os comandos SQL.

1) TABELA DE PROFESSORES TB_PROFESSOR

ID_PROFESSOR	**NOME_PROFESSOR**
JOI	JOILSON CARDOSO
OSE	OSEAS SANTANA
VIT	VITOR VASCONCELOS
FER	JOSE ROBERTO FERROLI
LIM	VALMIR LIMA
EDS	EDSON SILVA
WAG	WAGNER OKIDA

2) TABELA DE ALUNOS TB_ALUNO

COD_ALUNO	NOME_ALUNO	END_ALUNO	SIGLA_ESTADO	ID_CLASSE
1	ANTONIO CARLOS PENTEADO	RUA X	SP	1
2	AUROMIR DA SILVA VALDEVINO	RUA W	SP	1
3	ANDRE COSTA	RUA T	SP	1
4	ROBERTO SOARES DE MENEZES	RUA BW	SP	2
5	DANIA	RUA CCC	SP	2
6	CARLOS MAGALHAES	AV SP	SP	2
7	MARCELO RAUBA	AV SAO LUIS	SP	3
8	FERNANDO	AV COUNTYR	SP	3
9	WALMIR BURIN	RUA SSISIS	SP	3

3) TABELA DE DICIPLINAS TB_DISCIPLINA

ID_DISCIPLINA	NOME_DISCIPLINA	ID_PROFESSOR_DISCIPLINA	NOTA_MINIMA_DISCIPLINA
MAT	MATEMATICA	JOI	7
POR	PORTUGUES	VIT	5
FIS	FISICA	OSE	3
HIS	HISTORIA	EDS	2
GEO	GEOGRAFIA	WAG	4
ING	INGLES	LIM	2

4) RELACIONAMENTO ALUNO X DISCIPLINA TB_ALUNO_DISCIPLINA

COD_ALUNO	ID_DISCIPLINA	NOTA_ALUNO
1	MAT	0
2	MAT	0
3	MAT	1
4	POR	2
5	POR	2
6	POR	2
7	FIS	3
8	FIS	3
9	FIS	3
1	POR	2
2	POR	2
7	POR	2
1	FIS	3

10.3 Exercícios de SELECT básico

1). Queremos selecionar todos os alunos cadastrados.

2). Queremos selecionar todos os nomes de disciplina, cujo a nota mínima seja maior que 5 (cinco).

3). Queremos selecionar todas disciplinas que tenham nota mínima entre 3 (três) e 5 (cinco).

10.4 Exercícios de SELECT (Ordenando e agrupando dados)

1). Queremos selecionar todos os alunos em ordem alfabética de nome de aluno, e também o número da classe que estuda.

2). Selecionaremos o item anterior, porém ordenado alfabeticamente pelo identificador do aluno de forma descendente (ascendente é “default”). .

3). Selecionaremos todos os alunos cursando as disciplinas de matemática e de português agrupados por aluno e disciplina.

10.5 Exercícios de SELECT (Junção de Tabelas)

1). Queremos selecionar todos os nomes de alunos que cursam Português ou Matemática.

2). Queremos selecionar todos os nomes de alunos cadastrados que cursam a disciplina FÍSICA e seus respectivos endereços .

3). Queremos selecionar todos os nomes de alunos cadastrados que cursam física e o andar que se encontra a classe dos mesmos. Preste atenção ao detalhe da concatenação de uma ***string*** "andar" junto à coluna do número do andar (Apenas para estética do resultado).

10.6 Exercícios de SELECT (OUTER JOIN)

1. Selecionar todos os Professores com suas respectivas disciplinas e os demais Professores que não lecionam disciplina alguma.

10.7 Exercícios de SELECT (USE Clausula IN e/ou SUBSelect).

Não pode usar junção.

2. Selecionar todos os nomes de professores que tenham ministrado disciplina para alunos que sejam do Estado do Piaui, cujo a classe tenha sido no terceiro andar.

OBS: *Após termino da query anterior, executar a mesma utilizando JUNÇAO (JOIN). Comparar os Planos de Execução de cada uma*.