2016

Banco de Dados

Professor Rodrigo Ramos Dornel

# [BANCO DE DADOS]

Conteúdo da disciplina de Banco de Dados

# Sumário

# 1. Introdução

A extração de informações a partir de processos ou rotinas de trabalho é de fundamental importância para a área de Sistemas de Informação. A revisão de processos, otimização e verificação de rotinas e resultados, não são possíveis de serem efetuadas sem que exista uma análise segura e consistente sobre uma massa de dados. Representar, extrair, manter, gerenciar e organizar esta massa de dados é o objetivo final da utilização de um sistema de Banco de Dados.

O domínio da informação continua sendo fator de vital importância para a sobrevivência em qualquer atividade do mercado atual. Porém, antes de armazenar e gerenciar a informação é necessário um processo de modelagem do ambiente, este processo de modelagem permitirá a retirada dos dados do ambiente. E a partir deste processo todas as outras etapas de trato da informação se seguirão. É evidente a importância do domínio desta tecnologia, e para que este domínio seja alcançado faz-se necessário um conhecimento da teoria de Banco de Dados.

Segundo DATE (2000), um sistema de banco de dados pode ser interpretado como um sistema computadorizado de armazenamento de registros. Este sistema não se resume apenas ao armazenamento dos registros, oferece também a possibilidade de manipulação e gerenciamento destes registros. A manipulação dos registros poderá acontecer sob a forma de alterações, inserções e exclusões de dados. De acordo com SILBERSCHATZ et al. (1999), um Sistema Gerenciador de Banco de Dados (SGBD) é constituído por um conjunto de dados integrados a um conjunto de programas para acesso a esses dados.

# 2. Níveis de Abstração dos Dados

A utilização de um SGBD permite que o usuário acesse os dados mantidos em arquivos de disco, sem a necessidade do conhecimento da manipulação destas estruturas de informações. Isto é possível tendo em vista a utilização, pelo sistema, de níveis de abstração no acesso aos dados. Segundo SILBERSCHATZ et al. (1999), os sistemas gerenciadores de banco de dados empregam três níveis de abstração no acesso aos dados: nível físico, nível lógico e nível de visão.

- Nível Físico: este nível descreve a forma de armazenamento dos dados, a granularidade pode chegar a bytes ou páginas de dados.
- Nível Lógico: este nível descreve quais dados estão armazenados, descreve ainda os relacionamentos entre estes dados. Este nível é amplamente utilizado pelos administradores de bancos de dados.
- Nível de Visão: é através deste nível que se fornece ao usuário final a possibilidade de visualização das informações mantidas no banco de dados. Neste nível é possível de se limitar o acesso ao banco de dados, liberando a visualização de apenas uma parte dos dados aos usuários.

# 3. Modelos de Dados

Os dados podem ser armazenados sob vários modelos ao nível lógico de dados: modelo relacional, modelo de rede, modelo hierárquico e o modelo orientado a objetos. O foco deste tópico é apresentar o conceito de banco de dados de acordo com o modelo relacional.

## 3.1 Modelo Relacional

Neste modelo utilizam-se tabelas para a representação dos dados, os relacionamentos entre os dados são também representados por tabelas. As tabelas possuem várias colunas (atributos ou campos), e várias linhas para o armazenamento dos registros (tuplas). A representação dos dados relativos a clientes em um banco de dados sob o modelo relacional poderia ser descrita conforme a tabela abaixo:

| Codigo_cliente | Nome_cliente | Sexo_cliente | Nasc_cliente | Fone_cliente |
|---|---|---|---|---|
| A101 | José da Silva | M | 10/01/1985 | 444-1111 |
| A102 | Maria da Silva | F | 12/05/1987 | 555-2222 |
| A103 | Juliana de Moraes | F | 10/10/1989 | 444-2222 |

### 3.1.1 Estrutura dos bancos de dados relacionais

Um banco de dados relacional estrutura-se basicamente, sobre uma coleção de tabelas. As tabelas que constituem um banco de dados relacional são compostas de colunas e linhas, cada linha da tabela representa um relacionamento entre um conjunto de valores.

A tabela a seguir servirá como base para uma discussão um pouco mais aprofundada dos conceitos da estrutura de um banco de dados relacional.

| Nome_agencia | Numero_conta | saldo |
|---|---|---|
| Joinville | C-100 | 500 |
| Blumenau | C-200 | 800 |
| Beira Mar | C-250 | 400 |
| Universitária | C-300 | 300 |
| Criciúma | C-400 | 900 |
| Verde Vale | C-800 | 550 |
| Cidade das Flores | C-900 | 1000 |

Relação ***Conta***

A tabela acima será identificada como tabela ou relação conta, possui três colunas: nome_agencia, numero_conta e saldo. As colunas serão referenciadas como atributos, o conjunto de valores permitidos para cada atributo (coluna) representará o domínio do atributo. O domínio do atributo nome_agencia é o conjunto de todos os nomes possíveis e permitidos para as agências, poderemos atribuir a denominação D1 para este conjunto. O conjunto dos valores possíveis do atributo numero_conta iremos identificar como D2, e como D3 o conjunto de valores possíveis e permitidos para o atributo saldo.

Observando a tabela (relação) anterior, podemos notar que qualquer linha da tabela consiste de uma 3-tupla (v1,v2,v3) onde v1 representa o nome de uma agência, v2 representa um número de conta e v3 um valor de saldo. Ou ainda : v1 está no domínio D1, v2 está no domínio D2 e v3 está no domínio D3. Podemos também afirmar que a tabela (relação) conta é um subconjunto de D1 x D2 x D3. Uma tabela de n atributos será um subconjunto de D1 x D2 x ... x Dn-1 x Dn.

Da matemática podemos buscar uma definição para relação como sendo um subconjunto de um produto cartesiano de uma lista de domínios. Tendo em vista que podemos considerar as tabelas como relações, podemos usar os termos matemáticos de relação e tuplas significando respectivamente tabela e registros. Podemos pensar em tabelas (relação) como sendo um conjunto de tuplas. Na relação conta podemos encontrar sete tuplas, iremos assumir a seguinte notação para fazermos referencia às tuplas: t[1] serão os valores da tupla para o primeiro atributo, t[2] os valores da tupla para o segundo atributo e assim consecutivamente.

### 3.1.2 Esquema de banco de dados

O esquema de banco de dados nada mais é que o seu esquema lógico, suas definições de estrutura. Uma instância de banco de dados pode ser imaginada como uma posição do banco de dados em um instante de tempo. Aproveitaremos a relação conta para mostrar a nomenclatura que será adotada a partir de agora:

Esquema da relação conta : Esquema_conta = (nome_agencia, numero_conta, saldo

*especifica a estrutura da relação*

Agora, o fato de conta ser uma relação em Esquema_conta será expresso por

Conta(Esquema_conta) especifica os valores assumidos pela relação

Frequentemente utilizaremos o termo relação significando instancia da relação, para auxiliar na compreensão do tema vamos apresentar todas as relações que serão utilizadas neste exemplo em estudo.

| Nome_agencia | Cidade_agencia | fundos |
|---|---|---|
| Verde Vale | Blumenau | 900000 |
| Cidade das Flores | Joinville | 800000 |
| Universitária | Florianópolis | 750000 |
| Joinville | Joinville | 950000 |
| Beira Mar | Florianópolis | 600000 |
| Criciúma | Criciúma | 500000 |
| Blumenau | Blumenau | 1100000 |
| Germânia | Blumenau | 400000 |

Relação ***Agencia***

Esquema_agencia = (nome_agencia, cidade_agencia, fundos)

| Nome_cliente | Rua_cliente | Cidade_cliente |
|---|---|---|
| Ana | XV de Novembro | Joinville |
| Laura | 07 de Setembro | Blumenau |
| Vânia | 01 de Maio | Blumenau |
| Franco | Felipe Schmidt | Florianópolis |
| Eduardo | Beira Mar Norte | Florianópolis |
| Bruno | 24 de maio | Criciúma |
| Rodrigo | 06 de agosto | Joinville |
| Ricardo | João Colin | Joinville |
| Alexandre | Margem esquerda | Blumenau |
| Luciana | Estreito | Florianópolis |
| Juliana | Ririu | Joinville |

Relação ***cliente***

Esquema_cliente = (nome_cliente, rua_cliente, cidade_cliente)

| Nome_cliente | Numero_conta |
|---|---|
| Ana | C-100 |
| Laura | C-200 |
| Eduardo | C-250 |
| Bruno | C-400 |
| Rodrigo | C-900 |
| Vânia | C-800 |
| Luciana | C-300 |

Relação ***depositante***

Esquema_depositante = (nome_cliente, numero_conta)

| Nome_agencia | Numero_empréstimo | Total |
|---|---|---|
| Joinville | L-10 | 2000 |
| Blumenau | L-20 | 1500 |
| Beira Mar | L-15 | 1800 |
| Criciúma | L-30 | 2500 |
| Cidade das Flores | L-40 | 3000 |
| Verde Vale | L-35 | 2800 |
| Universitária | L-50 | 2300 |

Relação ***empréstimo***

Esquema_empréstimo = (nome_agencia, numero_empréstimo, total)

| Nome_cliente | Numero_empréstimo |
|---|---|
| Ana | L-10 |
| Laura | L-20 |
| Eduardo | L-15 |
| Bruno | L-30 |
| Rodrigo | L-40 |
| Vânia | L-35 |
| Luciana | L-50 |

Relação ***devedor***

Esquema_devedor = (nome_cliente, numero_empréstimo)

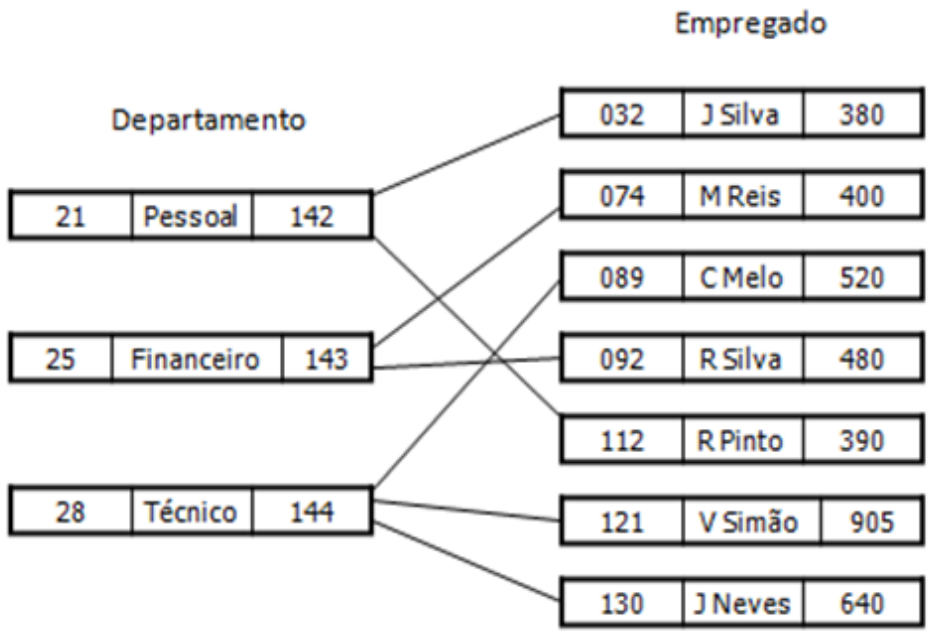
Voltando o foco para a relação agencia, notamos que o atributo nome_agencia aparece também na relação conta. Esta redundância de informações nos permite interligar ou, relacionar

tuplas de tabelas (relações) distintas. Este nível de redundância de informações nos é interessante, tendo em vista de que através destes atributos podemos efetuar relacionamentos entre várias tabelas (relações).

Poderíamos imaginar o armazenamento de todas as informações em uma única relação. Porém, neste caso, a redundância de dados seria muito alta, o que sem dúvida alguma iria contribuir com um alto custo de processamento nas atividades de atualização de informações.

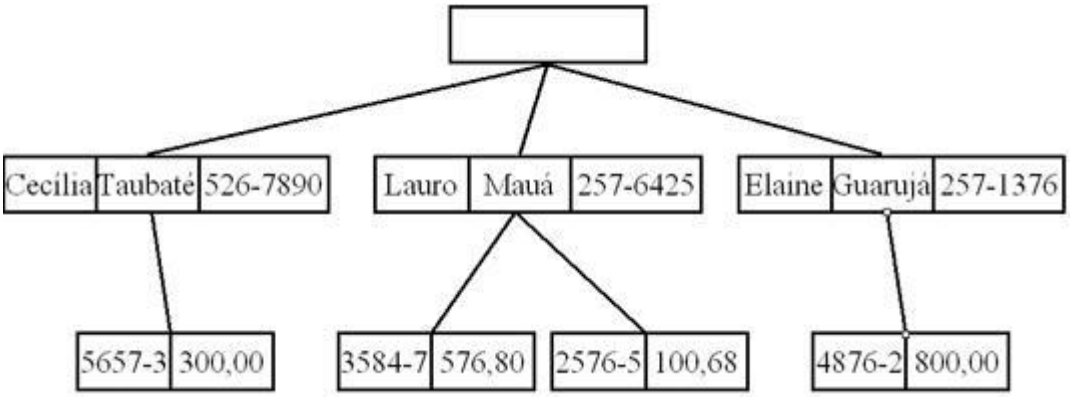
### 3.2 Modelo de Rede

Neste tipo de modelo os dados são representados por um conjunto de registros e os relacionamentos são representados por ligações entre estes registros, a figura dos ponteiros se faz presente neste tipo de modelo.




### 3.3 Modelo Hierárquico

Similar ao modelo em rede, apenas com a diferença de que os registros são organizados em árvores hierárquicas, ao invés de gráficos como no modelo em rede.




### 3.4 Modelo Orientado a Objeto

Este modelo surgiu na década de 80 na tentativa de oferecer uma capacidade de armazenamento específica que o modelo relacional não atendia. Hoje em dia os armazenamentos de informações de sistemas de informações geográficas e os sistemas CAD são as maiores aplicações deste conceito. Os conceitos de classes, instâncias e métodos são aplicados neste modelo de dados.

| Modelo Relacional | Modelo OO |
|---|---|
| Tabelas (entidades) | Objetos |
| Linhas (registros) | Tuplas |
| Query's(consultas,etc) | Native Query's |
| Sql Ansci | Métodos, construtores |

```
Q15: (select struct (sobrenome : s.nome.unome, primeiro_nome:
                     s.nome.pnome, mdc: s.mdc)
      from s in departamentocc.forma_especialistas order by mdc desc) [0:2];
```

### 3.5 Modelos não estruturados

XML

# 4. Arquitetura de Banco de Dados

A arquitetura de um sistema de banco de dados é totalmente dependente pelo sistema computacional sobre o qual o banco de dados está sendo executado. De acordo com estes sistemas computacionais, é possível identificar algumas arquiteturas de bancos de dados: centralizado, cliente-servidor, paralelos e distribuídos.

## 4.1 Centralizados

Sistemas de banco de dados centralizados são aqueles executados sobre um único sistema computacional que não interagem com outros sistemas. Um sistema centralizado consiste basicamente de algumas CPUs e dispositivos de controle conectados através de um barramento comum de modo a proporcionar acesso à memória compartilhada. As CPUs possuem memórias cache locais no intuito de fornecer acesso rápido aos dados através do armazenamento local de algumas informações. Os dispositivos de controle atendem a tipos específicos de dispositivos. Tendo em vista o fato de que a memória é centralizada, poderá existir alguma contenção no acesso concorrente à área de memória, acesso efetuado tanto por parte das CPUs quanto pelos dispositivos de controle. A memória cache exerce um papel fundamental na redução deste processo de contenção, pois permite que algumas informações possam ser alcançadas através da utilização da memória cache.

Um sistema monousuário é tipicamente constituído por uma unidade de trabalho, com uma única CPU e um ou dois discos rígidos, com um sistema operacional capaz de oferecer suporte a apenas um único usuário. Um sistema multiusuário possui um número maior de discos e área de memória, pode contar com várias CPUs e um sistema operacional multiusuário. Atende a um grande número de usuários que estão conectados ao sistema por meio de terminais, estes sistemas são conhecidos como sistemas servidores.

Os sistemas de banco de dados monousuários, na maior parte das vezes não oferece o tratamento da concorrência, visto que o acesso ocorre somente por um usuário. Da mesma forma que não oferece o tratamento da concorrência, estes sistemas não dispõem de mecanismos de recuperação de falhas, tampouco oferecem o suporte à linguagem SQL. Os sistemas de computadores de propósito geral possuem atualmente múltiplos processadores, eles possuem paralelismo de granulação-grossa, com um número limitado de processadores, todos compartilhando a memória principal. Os bancos de dados rodando nestes equipamentos não promovem o particionamento de uma consulta entre os processadores: ao contrário, cada consulta é executada em um único processador, permitindo que diversas consultas sejam executadas concorrentemente. Estes sistemas permitem a obtenção de um alto grau de throughput (número de transações executadas por segundo), porém isto não implica em diminuição no tempo de processamento de uma única transação, pois a mesma (transação) não pode ser fragmentada para execução concorrente em vários processadores.

Bancos de dados processados em equipamentos de um único processador, já dispõem de recursos multitarefas, permitindo que diversos processos sejam executados em um mesmo processador de modo compartilhado, a velocidade deste compartilhamento dá ao usuário a sensação de que estes processos estão sendo executados em paralelo. Desta forma, os equipamentos com paralelismo de granulação-grossa parecem idênticos a um equipamento de um único processador. Os equipamentos de granulação-fina possuem um grande número de processadores, os sistemas de bancos de dados rodando nesse tipo de equipamento podem processar consultas submetidas pelos usuários em paralelo.

## 4.2 Cliente-Servidor

Os terminais conectados em sistemas centralizados estão sendo substituídos por computadores pessoais, as interfaces estão sendo remodeladas para o trato com os computadores pessoais. Desta forma, os sistemas centralizados agem como sistemas servidores que atendem a solicitações de sistemas clientes. As atividades de um sistema de banco de dados podem ser divididas em duas categorias: front-end e back-end. O lado back-end engloba o gerenciamento de acesso, desenvolvimento e otimização de consultas, controle de concorrência e os processos de recuperação de falhas. No lado front-end encontram-se as ferramentas responsáveis pelos formulários, relatórios e recursos de interface gráfica. A linguagem SQL atua na interface entre o back-end e o front-end. Desta maneira, os sistemas servidores podem ser caracterizados como servidores de transações (query-server) e servidores de dados.

**Servidores de transações:** nos sistemas centralizados, o front-end e o back-end encontram-se em atuação dentro do mesmo sistema. Já os servidores de transações atuam no lado back-end da arquitetura, enquanto que os computadores pessoais atuam como clientes do servidor. Os clientes enviam solicitações ao sistema servidor no qual essas transações são executadas e os resultados são enviados de volta ao cliente que tem a responsabilidade de exibir esses dados. Programas de aplicação de interface possibilitam aos clientes a construção de comandos SQL para envio e execução no servidor, a ODBC (Open Database Connectivity) é um exemplo de um programa de interface. Interfaces cliente-servidor não ODBC são também utilizadas em sistemas de processamento de transações, nestes casos acontece a utilização das chamadas de procedimento transacional remota. Todas as chamadas de procedimentos remotas feitas pelo cliente são englobadas em uma única transação para o servidor. Desta forma, no caso de aborto da transação, o servidor poderá reverter os efeitos da chamada remota, garantindo então a propriedade da atomicidade (ACID).

**Servidores de dados:** os sistemas servidores de dados têm utilização em redes locais, onde existem conexões de alta velocidade entre clientes e servidores, a capacidade de processamento dos clientes é comparável com a capacidade de equipamentos servidores. Neste ambiente, acontece o envio dos dados do servidor para que o processamento aconteça no lado

cliente, e o posterior envio dos resultados para o servidor. Este tipo de arquitetura tem sido utilizada em sistemas de bancos de dados orientados a objetos. A unidade de transferência para os dados pode ser de granularidade grossa, como uma página, ou de granularidade fina, como uma tupla. Se a unidade de comunicação for um único item, o custo para a troca de mensagens é alto se for comparado ao volume de dados transmitido. Desta forma, quando um item é solicitado, faz sentido enviar outros itens que possam ser utilizados, esta busca antecipada é conhecida como prefetching. A transferência de uma página pode ser considerada uma forma de prefetching, pois os outros itens da página de dados serão também enviados na mesma solicitação. Visto que acontece o trânsito dos dados entre os clientes e o servidor, é necessário o estabelecimento de bloqueios nestes itens de dados. A utilização de páginas para a transferência de dados incorre na emissão do bloqueio da página na totalidade, diminuindo a concorrência no acesso pelos dados mantidos em uma mesma página de dados. Os dados que navegam para um cliente durante uma transação podem ser armazenados na memória do equipamento cliente, mesmo após a utilização destes itens pelo cliente. Este procedimento aumenta a velocidade na busca pela informação, pois evita a emissão de mais uma requisição ao servidor. No entanto, este procedimento requer um cuidado especial, que é a verificação da validade daquela informação armazenada localmente, abrindo a possibilidade do trabalho com informação desatualizada.

### 4.3 Sistemas Paralelos

Os sistemas paralelos permitem imprimir uma maior velocidade ao processamento e às operações de I/O, através do uso paralelo das diversas CPU's e discos. Neste tipo de arquitetura, várias operações são realizadas simultaneamente, ao contrário do processamento serial, onde os passos do processamento são sucessivos. Os equipamentos de granulação-grossa consistem de poucos e poderosos processadores, já os de granulação-fina utilizam milhares de pequenos processadores. Existem duas medidas principais para a avaliação do desempenho de um sistema de banco de dados: o throughput, que representa o número de tarefas que podem ser realizadas em um intervalo de tempo, e o tempo de resposta que mede o tempo gasto para o sistema processar uma única tarefa. A arquitetura paralela pode ser divida ainda nas seguintes arquiteturas de bancos de dados paralelos: memória compartilhada, disco compartilhado, ausência de compartilhamento e hierárquico.

1. **Memória compartilhada:** na arquitetura paralela com memória compartilhada, os processadores e discos acessam a memória comum por meio de cabo ou alguma rede de interconexão. A vantagem da utilização da memória compartilhada é a eficiência na comunicação entre os processadores. Esta comunicação é feita através da troca de mensagens utilizando a memória. O grande problema deste tipo de arquitetura é no uso de mais de 32 ou 64 processadores, o bus ou a interconexão de rede torna-se um gargalo do sistema, pois é compartilhado por todos os processadores. A partir de um determinado ponto, a inclusão de mais

processadores não traz ganho para o sistema, visto que passarão a maior parte do tempo esperando por um momento de utilização do bus para o acesso à memória.

2. **Disco compartilhado**: neste modelo, todos os processadores podem ter acesso direto a todos os discos, através de interconexão por rede, sendo que os processadores possuem memórias próprias. Visto que cada processador possui memória própria, nesta arquitetura não existirá o gargalo no bus para acesso à memória. Ainda neste modelo, a tolerância a falhas tem um ganho razoável, pois mesmo que ocorra uma falha no processador, ou na sua memória, outro processador poderá assumir estas tarefas e acessar o banco que continua residindo nos discos compartilhados. Existe ainda a possibilidade de através do RAID, oferecer uma tolerância à falhas para o subsistema de disco. Sistemas construídos sobre esta arquitetura são conhecidos como clusters. O grande problema desta arquitetura é na interconexão com o subsistema de discos, visto que a velocidade de comunicação em acesso à disco é infinitamente menor que no tráfego de informações através da memória compartilhada.

3. **Ausência de compartilhamento**: na ausência de compartilhamento, cada equipamento consiste em um processador, uma memória e discos. Os processadores dos nós podem se comunicar utilizando uma rede de alta velocidade. Cada nó tem a função de servidor dos dados mantidos nos seus discos. Desta forma, apenas as requisições para dados mantidos em outros nós irão sofrer com a queda de desempenho das operações de I/O executadas nos nós remotos.

4. **Hierárquica**: esta arquitetura combina os compartilhamentos de memória e discos e o modelo sem compartilhamento. No nível mais alto, o sistema constitui-se de nós conectados por uma rede sem compartilhar discos ou memória entre eles. No topo da linha existe então, uma arquitetura sem compartilhamento. Cada nó do sistema pode ser um sistema com memória compartilhada entre alguns processadores, ou ainda, cada nó pode ser um subsistema de discos compartilhados e cada um desses sistemas que compartilham um conjunto de discos poderiam também compartilhar memória.

### 4.4 Sistemas distribuídos

Neste tipo de arquitetura não existe o compartilhamento de memória ou discos, o banco de dados é distribuído sobre uma série de sistemas de bancos de dados localizados em nós de uma rede de comunicação, estes nós são conhecidos como sites. Nos sistemas distribuídos as

transações podem ser classificadas em: locais e globais. As transações locais acessam um único site, aquele onde a transação teve o seu início, já as transações globais possuem a característica de buscar informações em diversos sites.

Em um sistema de banco de dados distribuído, um dos grandes objetivos é armazenar os dados nas proximidades do usuário final, permitindo desta forma que o custo do acesso aos dados por aquele usuário seja resumido, sem a necessidade de envolver a comunicação com outros sites do ambiente.

O grau de autonomia que se obtêm com esta arquitetura é uma consequência da distribuição dos dados, cada site tem a sua própria administração do sistema de banco de dados. Esta administração leva em conta características próprias da localização daquele site, a distribuição dos dados permite uma maior independ6encia na sua administração.

A disponibilidade do sistema sofre um acréscimo considerável sob esta arquitetura, a inoperância de um nó não necessariamente impacta na indisponibilidade de todo o sistema. A possibilidade da utilização da replicação de dados aumenta a tolerância à falhas neste modelo, pois os dados que eram mantidos em um nó falho poderão ser acessados, caso estejam replicados, em outros sites do ambiente distribuído.

## 4.5 Computação móvel

A utilização de bancos de dados em ambientes de computação móvel tem crescido bastante nos últimos tempos. A possibilidade da mobilidade dos usuários enquanto executa-se a consulta ao banco de dados, tem aberto uma série de perspectivas de aplicações desta tecnologia. Sob a tecnologia da computação móvel, algumas arquiteturas de bancos de dados podem ser apresentadas: cliente-servidor, par-par e agentes móveis.

A tradicional arquitetura cliente-servidor necessita sofrer alguns ajustes para que possa atender o ambiente da computação móvel na totalidade, neste sentido surgem duas modificações desta arquitetura, que serão tratadas a seguir.

### 4.5.1 Cliente-agente-servidor

Esta arquitetura sugere a inclusão de um agente que represente o cliente na rede fixa, proporcionando desta forma a migração de carga de processamento da unidade móvel para o servidor de dados da rede estacionária. Nesta arquitetura, a unidade móvel pode emitir, através do agente, as requisições que necessitar e entrar em modo de espera (doze). A responsabilidade de alcançar as respostas para as requisições da unidade móvel fica a cargo do agente. No momento da reativação da unidade móvel o agente será encarregado de enviar as informações requisitas pela unidade móvel. No entanto, esta arquitetura ainda não oferece o suporte para o trabalho desconectado, quando a unidade móvel perder a conexão com a rede estacionária, as

requisições direcionadas para a estação cliente serão enfileiradas pelo agente localizado na rede estacionária e, após a reconexão serão enviadas para a unidade móvel.

### 4.5.2 Cliente-interceptador-servidor

Esta arquitetura resolve a impossibilidade do trabalho desconectado da arquitetura cliente-agente-servidor. A figura do agente foi "replicada" para o lado cliente (agente-lado-cliente) além do lado servidor (agente-lado-servidor). Desta forma, os agentes interceptam as requisições clientes e servidoras trabalhando de forma cooperativa no atendimento de cada requisição. A possibilidade do trabalho desconectado vem do fato de se poder fazer uso de um cache no agente-lado-cliente. Durante o período da desconexão, as unidades móveis terão suas requisições atendidas através da utilização do cache mantido pelo agente-lado-cliente. Caso a informação não esteja disponível no cache da unidade móvel, a requisição será enfileirada pelo agente-lado-cliente e no momento da reconexão será entregue aos cuidados do agente-lado-servidor.

### 4.5.3 Arquitetura par-par

Nesta arquitetura não existe a distinção entre as figuras de servidores e clientes, cada estação tem a funcionalidade total tanto de servidor quanto de estação cliente. Neste caso a desconexão revela-se em um grande problema, pois a indisponibilidade de um site pode comprometer uma transação por completo. No intuito de amenizar este problema, a introdução de agentes para agirem como representantes das unidades pode ser uma saída.

# 5. Linguagens de Banco de Dados

## 5.1 Linguagem de Manipulação dos Dados

DML
Acesso e manipulação do dados
Recuperação, inserção, exclusão e modificação.

Linguagem de Manipulação dos Dados

- DMLs procedurais
- DMLs declarativas

DMLs procedurais

- Necessária especificação dos dados a serem acessados e como obtê-los, mais complexas.
- Blocos de execução e execução remota.

DMLs declarativas

- O usuário apenas especifica que dados quer, sem se preocupar com a estrutura.
- SELECT * FROM TABELA

## 5.2 Linguagens de Definição dos Dados

- Definição de um esquema de banco de dados e especificar propriedades adicionais dos dados.
- Esta linguagem define questões de armazenamento dos dados, definição dos dados e mais importante, definem restrições de consistência.
- Restrições de Domínio: um domínio possível de valores precisa ser associado com cada atributo. As restrições de domínio são a forma mais elementar de restrição de integridade.
- Integridade referencial: precisamos garantir que um determinado valor que aparece em uma determinada relação também apareça em outro determinado conjunto.
- Integridade referencial: as vezes uma modificação pode tentar violar esta restrição, portanto o procedimento e de reverter essa alteração.
- Assertiva: muitas regras precisam ser validadas em um banco de dados, e uma delas pode representar um conjunto de condições verdadeiras que permite uma execução.
- Assertiva
- “Todo empréstimo tem pelo menos um cliente que mantém uma conta com um saldo mínimo de $1.000”
- Autorização: define o nível de acesso de cada usuário aos dados contidos no banco de dados.

- Autorização de leitura
- Atualização
- Exclusão
- Dicionário de dados e Metadados.
- Tabelas do sistema.
- Tabelas de documentação do sistema.
-

# 6. Modelo Entidade Relacionamento

O Modelo Entidade – Relacionamento (MER) foi concebido com a intenção de oferecer um mecanismo de representação dos ambientes reais para um modelo de dados. Os objetos do mundo real e os relacionamentos existentes entre eles podem ser representados sob a forma de um banco de dados, utilizando-se dos conceitos do Modelo Entidade – Relacionamento (MER).

## 6.1 Conjunto de Entidades

No Modelo Entidade-Relacionamento, as entidades irão identificar e representar os objetos do mundo real, podem tanto ser concreto como abstratas. Um conjunto de entidades é um conjunto que abrange entidades de mesmo tipo que compartilham as mesmas propriedades: os atributos. Na modelagem de um banco de dados para o armazenamento de informações relativas às atividades de uma empresa, os clientes podem ser representados através de uma entidade chamada cliente. Neste mesmo ambiente poderemos identificar a entidade denominada funcionários, representando as informações relativas aos funcionários daquela empresa.

Cada objeto que possa ser identificado no mundo real, possui uma série de características próprias que o identificam e o diferenciam com relação aos demais objetos. O objeto funcionário pode ter o número de filhos como uma característica, poderá ainda possuir uma matrícula que o identifique, com toda a certeza possui um nome e uma data de nascimento. Todas estas informações que identificam e de certa forma, constroem o objeto, são conhecidas como atributos do objeto. Por consequência, todas as entidades possuem uma série de atributos que oferecem informações sobre o objeto que está representando. Desta forma, podemos pensar em uma entidade como um conjunto de atributos.

É lógico supor que cada atributo possui uma série de valores possíveis de serem armazenados para aquela característica específica daquele objeto. Esta faixa de valores possíveis de serem armazenados sob aquele atributo é conhecida como domínio. No exemplo da entidade funcionário, podemos especificar uma série de valores possíveis de serem armazenados no atributo data de nascimento, certamente seria lógico supor que apenas os valores que representem uma data válida, e nunca maior que a data da inserção do valor (ninguém nasce amanhã, todos já nasceram em algum instante de tempo anterior) seria o domínio do atributo data de nascimento.

Podemos representar a entidade **funcionário** através de uma série de atributos:

**Funcionário** (nome, data de nascimento, matricula, cpf, identidade, sexo, nr filhos, fone, endereço, bairro, cidade, cep)

A seguir apresentaremos uma classificação dos atributos:

- **Simples:** neste grupo estão classificados os atributos que não podem ser divididos. Ex: sexo_funcionário

- **Compostos:** neste caso os atributos que podem sofrer algum tipo de divisão de forma a serem identificados novos atributos como resultado desta divisão. Ex: endereço_cliente pode ser divido em rua_cliente, nr_cliente, cidade_cliente, cep_cliente.
- **Monovalorados:** neste grupo estarão classificados os atributos que aceitam apenas valores simples para uma entidade. Ex: matricula_funcionário.
- **Multivalorados:** neste caso os atributos aceitam um conjunto de valores para cada instância da entidade. Ex: fone_cliente, o cliente pode ter vários números telefônicos, desta forma é possível a atribuição de uma série de valores para um mesmo cliente.
- **Nulos:** como seria lógico supor, neste caso estarão representados os atributos que aceitam a ocorrência de valores nulos. Ex: email_cliente, nem todos os clientes possuem um endereço eletrônico.

As entidades podem ser classificadas em **regulares** e **fracas.** As entidades fracas são aquelas que dependem de outra entidade para a sua própria existência. Podemos imaginar uma entidade chamada **dependentes** totalmente dependente da entidade **funcionários**. Desta forma, a entidade **dependentes** é considerada uma entidade fraca, pois necessita que a entidade **funcionários** exista para que possa ser representada ( só existe dependente de algum funcionário, todo dependente tem relação com algum funcionário, não necessariamente todo funcionário possui algum dependente !!! ).

### 6.2 Relacionamentos

O **relacionamento** pode ser entendido como uma associação entre entidades, no MER iremos identificar e representar um relacionamento através de uma entidade. Utilizando as entidades **funcionários** e **dependentes,** poderíamos pensar em uma associação entre estas duas entidades de tal forma a representar o vínculo existente entre os funcionários e os seus dependentes. Desta maneira estaremos trabalhando com uma nova entidade chamada **func_depen,** que irá representar a associação e o relacionamento existentes entre as entidades **funcionários** e **dependentes.** As entidades que estão envolvidas em um relacionamento são denominadas de **participantes** do relacionamento, o número de participantes do relacionamento é denominado de **grau** do relacionamento. Na maior parte das vezes, os relacionamentos que existirão no modelo relacional serão do tipo binários: envolvendo duas entidades (ex: **func_depen** que relaciona o funcionário com o seu dependente). Existem, porém relacionamentos com grau maior do que 2 (binários), poderemos dar um exemplo de relacionamento ternário quando fizermos a representação dos funcionários, dependentes e o plano de saúde relativo a esta dependência. Desta forma teremos neste conjunto de relacionamentos atributos que identificam o funcionário (código do funcionário), o dependente (código do dependente) e o plano de saúde (código do plano).

O relacionamento é dito **total** se toda a instância de uma entidade participa de pelo menos uma ocorrência do relacionamento. No exemplo anterior pode-se afirmar com clareza que a participação de **dependentes** no relacionamento **func_depen** é **total**. O mesmo não ocorre em relação à participação de **funcionários,** pois não é verdade que todos os funcionários tenham que ter relação com algum dependente, neste caso a participação é dita **parcial.**

### 6.3 Cardinalidades

A cardinalidade informa o número de ocorrências de entidades às quais uma entidade pode estar associada através de um relacionamento. Assumindo duas entidades **A** e **B**, as seguintes situações de cardinalidade são possíveis de ocorrência:

**Um para um:** uma ocorrência de **A** estará associada a apenas uma única ocorrência de **B,** e uma ocorrência de **B** estará associada a uma única ocorrência de **A.**

**Um para muitos:** uma ocorrência de **A** estará associada a várias ocorrências de **B**, no entanto, uma ocorrência de **B** estará associada a uma única ocorrência de **A.**

**Muitos para muitos:** uma ocorrência de **A** poderá estar associada a muitas ocorrências de **B,** da mesma forma, uma ocorrência de **B** poderá estar associada a muitas ocorrências de **A .**

A cardinalidade pode variar conforme o modelo que se está considerando. No exemplo utilizado até agora podemos verificar a variação da cardinalidade existente entre as entidades **funcionários** e **departamentos**, vamos analisar 02 situações possíveis:

**1 Um funcionário pode estar vinculado a apenas um departamento:** neste caso, para uma ocorrência de funcionário existirá apenas uma ocorrência possível de departamento (**um**), entretanto, para uma única ocorrência de departamento poderemos ter várias ocorrências de funcionários (**muitos**). Neste caso a cardinalidade entre funcionários e departamentos é do tipo **muitos para um**. Geralmente, neste tipo de associação um atributo da entidade que identifique o lado "**um**" da associação é posicionado na entidade que representa o lado "**muitos**".

**2 Um funcionário pode estar vinculado a vários departamentos:** nesta situação uma ocorrência de funcionários pode ter associação com várias ocorrências de departamentos (**muitos**), e uma ocorrência de departamento pode estar associada com uma série de ocorrências de funcionários (**muitos**). Neste caso a cardinalidade é do tipo **muitos para muitos.** Os relacionamentos que possuem este grau de cardinalidade geralmente serão representados pela construção de uma nova entidade.

Fica claro nos exemplos anteriores que a cardinalidade depende da regra do negócio que está sendo analisado, no entanto, por se tratar de uma alteração no relacionamento entre as entidades, é fundamental de que se faça esta análise com um profundo conhecimento do ambiente que está sendo modelado.

### 6.4 Dependência de Existência

A dependência de existência é um tipo de restrição que pode ser representada na etapa da modelagem dos dados. Esta restrição identifica as entidades que apresentam algum tipo de dependência entre elas. Vamos supor que para a existência da entidade A, seja necessária a existência da entidade B, neste caso a entidade A será denominada de **entidade subordinada** e a entidade B será a **entidade dominante**. Podemos identificar estas entidades no exemplo de

**funcionários** e **dependentes**, a entidade **dependentes** é subordinada da entidade **funcionários** que neste caso, será considerada como a entidade dominante. Este tipo de restrição não permite que uma ocorrência da entidade dominante seja excluída sem a respectiva exclusão da ocorrência da entidade subordinada. Da mesma forma, não é possível que existam ocorrências de entidade subordinada sem que existam ocorrências da entidade dominante.

## 6.5 Chaves

Este é um tópico de fundamental importância para o trabalho com sistemas de bancos de dados, a utilização correta das chaves contribui para a manutenção da integridade do banco de dados. As restrições de integridade do tipo **identidade** e **referencial** são mantidas basicamente pela utilização dos vários tipos de chaves.

### 6.5.1 Chaves Candidatas

As chaves candidatas são um conjunto de atributos de uma determinada entidade que garantem a identificação única de uma ocorrência daquela entidade. Vamos imaginar uma série de atributos **K** de uma determinada entidade **E**, **K** será uma **chave candidata** para a entidade **E**, caso obedeça as duas condições a seguir:

- **Unicidade:** não existirão duas tuplas diferentes em **E** com o mesmo valor de **K.**
- **Irredutibilidade:** nenhum subconjunto de **K** poderá ter a propriedade de unicidade.

Exemplo:
23 – CESPE - Considerando-se o esquema e as dependências funcionais a seguir, é correto afirmar que o par composto
por matricula e nome é uma chave candidata.
Esquema_alunos = (matricula, nome, curso, telefone)
matricula, nome -> curso, telefone
matricula -> nome, curso, telefone
Resposta: Não poderá haver um subconjunto de atributos que formam a chave. Pela questão vemos que {matricula, nome} não formam uma chave candidata, pois {matricula} é um subconjunto de {matricula, nome} e possui a propriedade de unicidade. Sendo assim, {matricula, nome} não é irredutível.

No exemplo utilizado neste material vamos analisar o conjunto de atributos **cpf** e **nome** da entidade **funcionários**, este conjunto (cpf, nome) não pode ser considerado uma chave candidata, pois apesar de garantir que não existirão duas tuplas com os mesmos valores daquela combinação (cpf, nome) – propriedade da unicidade -, um subconjunto (cpf) da combinação inicial também apresentará a propriedade da unicidade. Desta forma esta combinação não atende os dois princípios necessários para a sua classificação como chave candidata. Já o conjunto dos atributos (nome, pai, mãe, dt_nascimento) poderá ser classificado como chave candidata, pois atende às duas propriedades necessárias.

### 6.5.2 Chave primária

A chave primária poderá ser escolhida a partir do conjunto de chaves candidatas possíveis para aquela entidade, da mesma forma que as chaves candidatas representam a identificação exclusiva das tuplas daquela entidade. É bom lembrar que uma chave primária representa um valor único e NÃO NULO**.**

### 6.5.3 Chave Estrangeira

Uma chave estrangeira é um conjunto de atributos de uma entidade **E1** cujos valores devem corresponder a valores de alguma chave candidata de outra entidade **E2.** No exemplo de **funcionários** e **departamentos**, para o caso em que um funcionário pode estar vinculado a apenas um departamento, o atributo codigo_departamento da entidade funcionário identifica a associação com departamento, este atributo é então considerado uma chave estrangeira para este relacionamento. As chaves estrangeiras são uma ferramenta poderosa para a manutenção da integridade referencial do banco de dados.

## 6.6 Diagrama Entidade - Relacionamento

A estrutura lógica do banco de dados pode ser expressa graficamente através do diagrama entidade-relacionamento, alguns dos componentes da construção deste diagrama serão apresentados a seguir:

**Retângulos:** representam os conjuntos de entidades, as entidades subordinadas serão representadas por um retângulo duplo;

**Elipses:** representam os atributos;

**Losangos:** representam os conjuntos de relacionamentos;

**Linhas:** unem os atributos aos conjuntos de entidades e os conjuntos de entidades aos relacionamentos;

**Elipses duplas:** representam atributos multivalorados;

**Losangos duplos:** representam o relacionamento entre uma entidade subordinada e sua entidade dominante;

**Linhas duplas:** indicam a participação total de uma entidade em um conjunto de relacionamentos.

Os atributos de um conjunto de relacionamentos que são membros da chave primária devem ser sublinhados. A **cardinalidade** do relacionamento pode ser indicada através de linhas com ou sem direcionamento ligando as entidades ou os relacionamentos. A seta apontará para o lado “um” da cardinalidade, caso a cardinalidade seja do tipo muitos para muitos, a linha não apresentará direcionamento algum.

## 6.7 Recursos do E-R

Apesar da possibilidade de representação da grande maioria dos bancos de dados, algumas vezes o modelo E-R não se adapta a determinadas situações do mundo real. Nestes casos surge a possibilidade da utilização de alguns recursos do E-R: **especialização, generalização, herança de atributos** e **agregação.**

### 6.7.1 Especialização

Um conjunto de entidades pode conter subconjuntos de entidades que são, de alguma forma, diferentes de outras entidades do conjunto. Este subconjunto pode apresentar atributos que não são compartilhados pelas demais entidades do conjunto. Um exemplo clássico desta situação é o caso da entidade "conta" para o banco de dados de uma instituição bancária, esta conta tanto pode ser uma conta corrente quanto uma conta poupança. A conta poupança apresenta um atributo que a conta corrente não possui: taxa de juros, já a conta corrente apresenta um atributo "limite do cheque especial" que a conta poupança não possui. A representação deste tipo de especialização acontece pela utilização de um triângulo, logo abaixo da entidade mais geral, rotulado com a palavra "ISA", a partir deste triângulo serão identificados os retângulos que identificam as entidades especializadas, com os respectivos atributos que lhes diferenciam.

### 6.7.2 Generalização

O refinamento do conjunto de entidades em níveis sucessivos de subgrupos indica um processo **top-down** de projeto, no qual as diferenciações são feitas de modo explícito. O projeto pode ser realizado de modo **bottom-up**, no qual vários conjuntos de entidades são resumidos em um conjunto de entidades de mais alto nível, com base nos atributos em comum. Este é o processo de **generalização**, que nada mais é do que uma **especialização** ao contrário. Qualquer um dos métodos escolhido (generalização ou especialização) apontará para um resultado final idêntico.

### 6.7.3 Herança de Atributos

A herança de atributos é uma consequência lógica do processo de especialização ou generalização. As entidades de nível inferior herdarão das entidades de nível superior uma série de atributos. Além de herdar alguns atributos, é implícita também a participação nos relacionamentos das entidades de nível superior.

## 6.8 Normalização de Dados

O objetivo principal da utilização de um banco de dados é a possibilidade de armazenamento dos dados de forma coerente, lógica e precisa. Na grande maioria das aplicações comerciais e, sobretudo nas aplicações do tipo OLTP (OnLine Transaction Processing) o nível de redundância de informação deve ser mantido baixo.

O armazenamento de informações repetidamente contribui para o uso desnecessário de espaço no banco de dados, bem como para uma carga maior de processamento em atividades de atualização de dados. A redundância é salutar (em um ambiente OLTP) quando serve de apoio no estabelecimento dos relacionamentos (chaves primárias e chaves estrangeiras) entre as entidades do banco de dados.

Em um ambiente do tipo OLAP (OnLine Analytical Processing) um bom nível de redundância de informações é um objetivo a ser alcançado. Neste ambiente a atividade de atualização dos dados é praticamente inexistente, operações de consultas prevalecem sobre as demais atividades. A possibilidade de manter um alto grau de redundância de informação otimiza o uso deste ambiente.

Independentemente do ambiente de trabalho do banco de dados (OLAP ou OLTP), a normalização de dados apresenta-se como um conceito amplamente utilizado para que se consiga uma boa modelagem do banco de dados. Vamos abordar neste curso apenas as formas de normalização 1FN, 2FN e 3FN. Nosso objetivo será buscar a normalização do ambiente para a 3FN.

O processo de normalização acontece em etapas distintas, na primeira delas buscamos identificar as entidades e atributos envolvidos no ambiente em estudo, nesta etapa deve-se definir uma chave para que se possa trabalhar com as formas normais. A partir daí aplicam-se sucessivamente as formas 1FN, 2FN e 3FN, abaixo iremos descrever as necessidades de cada forma normal.

### 6.8.1 Primeira Forma Normal

Na etapa da 1FN iremos retirar da entidade (relação) todos os atributos que possam contribuir para a repetição de registros a partir de uma única ocorrência de chave. Ou seja, para uma única ocorrência de valor da chave deve existir uma única ocorrência dos demais atributos.

### 6.8.2 Segunda Forma Normal

Nesta fase da normalização de dados iremos excluir da entidade (relação) todos os atributos que são dependentes de uma parte da chave composta, caso exista alguma chave composta na entidade (relação).

### 6.8.3 Terceira Forma Normal

Na terceira forma normal iremos retirar os atributos que são dependentes dos campos não-chave. A entidade estará na 3FN se estiver na 2FN e não possuir campos dependentes de atributos não-chave.

## 6.9 Álgebra Relacional

A álgebra relacional é uma linguagem de consultas que consiste de um conjunto de operações que tem como entrada uma ou duas relações e produz uma nova relação. As operações fundamentais na álgebra relacional são: select, project, union, set difference, cartesian product e rename. As operações fundamentais dão origem a outras operações: namely, set intersection, natural join, division e assignment.

### 6.9.1 Select $\sigma$

Seleciona tuplas (linhas, registros) que satisfaçam um predicado. A letra sigma é utilizada para simbolizar esta operação. O predicado (condição) estará subscrito a σ. O argumento da relação estará entre parênteses, seguindo o $\sigma$. A seguir alguns exemplos do operador select.

Selecionar todas as tuplas da relação agencia cuja agência esteja localizada na cidade de "Brooklyn":

$$\sigma_{\text{nome\_agência="Brooklyn"}} (\text{Agência})$$

Selecionar todas as tuplas da relação empréstimo cujas quantias são superiores a 1500:

$$\sigma_{\text{quantia} > 1500} (\text{Empréstimo})$$

### 6.9.2 Operadores Condicionais

Podemos utilizar nas operações de comparação operadores do tipo: $=, \neq, <, \leq, >, \geq$, além destes operadores poderemos combinar predicados através dos operadores lógicos e $(\vee)$ e $(\wedge)$ ou $(\neg)$, sendo OU, E e Negação. A seguir, alguns exemplos destas operações:

$$\sigma_{\text{nome\_agência="Brooklyn"} \wedge \text{quantia} > 1500} (\text{Empréstimo})$$

### 6.9.3 Project $\pi$

A operação project é uma operação primária e retorna o argumento da relação, exibindo apenas alguns atributos da relação. O resultado não permitirá tuplas duplicadas, visto tratar-se também de uma relação ou conjunto. Subscritos em $\pi$, estarão os atributos desejados no resultado. O argumento da relação continua vindo entre parênteses. Vamos então apresentar alguns exemplos desta operação:

Listar os nomes de todas as agências da relação conta:

$\pi$ nome_agência (Conta)

Selecionar os nomes das agências com saldo < 500:

$\pi$ nome_agência ($\sigma$ saldo < 500 (Conta) )

### 6.9.4 Union $\cup$

A operação union proporciona como resultado a união de duas relações. Na relação resultante não haverão tuplas duplicadas, pois a relação continua mantendo as características de um conjunto. Para que a operação union possa ser concretizada entre duas relações r e s ( r $\cup$s ), algumas condições devem ser satisfeitas:

a) as relações r e s devem possuir o mesmo número de atributos;

b) os domínios do i-ésimo atributo de r e o i-ésimo atributo de s devem ser os mesmos para todo i.

1) consulta que retorna os nomes dos clientes devedores: $\pi_{\text{nome\_cliente}}$ (tomador)

2) consulta que retorna os nomes dos clientes com conta: $\pi_{\text{nome\_cliente}}$ (depositante)

3) operação union entre os dois conjuntos anteriores: $\pi_{\text{nome\_cliente}}$ (tomador) $\cup$ $\pi_{\text{nome\_cliente}}$ (depositante)

### 6.9.5 Diferença entre conjuntos ( $^{-}$ )

A operação diferença entre conjuntos ( - ) traz como resultado as tuplas que estão em uma relação, mas não estão em outra. Ou seja, a expressão r – s expressa uma relação que contém as tuplas que estão em r mas não estão em s. Um exemplo desta operação poderia retornar o conjunto de resultados dos clientes que possuem contas mas não possuem nenhum empréstimo com o banco: π nome_cliente(depositante) – π nome_cliente (devedor). .Da mesma forma que na operação de union , as relações englobadas em uma operação de diferença entre conjuntos devem obedecer às regras mostradas na operação de union :

a) as relações r e s devem possuir o mesmo número de atributos;

b) os domínios do i-ésimo atributo de r e o i-ésimo atributo de s devem ser os mesmos para todo i.

1) consulta que retorna os nomes dos clientes devedores: $\pi_{\text{nome\_cliente}}$ (tomador)

2) consulta que retorna os nomes dos clientes com conta: $\pi_{\text{nome\_cliente}}$ (depositante)

3) operação diferença entre os dois conjuntos anteriores: $\pi_{\text{nome\_cliente}}$ (tomador) - $\pi_{\text{nome\_cliente}}$ (depositante)

### 6.9.6 Produto cartesiano ( X )

O produto cartesiano de duas relações R e S é o conjunto de tuplas onde cada tupla resultante é uma concatenação de uma tupla de R com uma tupla de S, para todas as tuplas de R e S. O número de elementos da relação resultante de um produto cartesiano entre duas relações R e S será o produto entre o número de elementos das relações R e S.

Exemplo 1 : Sejam duas relações R = { a, b, c } e S = { x, y } , a relação P = R x S será constituída pelo conjunto de valores P = {ax, ay, bx, by, cx, cy} , o número de elementos da relação P será 6 (3 x 2) elementos .

Exemplo 2 : Relacione os nomes de todos os devedores que tenham um empréstimo na agência Joinville.

$\sigma_{\text{nome\_agencia = "Joinville"}}$ **(devedor x emprestimo)** ( a )

Através da expressão acima, teremos uma relação com todas as combinações possíveis entre as tuplas de devedor e empréstimo, onde o atributo nome_agencia é igual a Joinville. Porém tendo em vista que a operação de produto cartesiano trabalha com todas as combinações possíveis entre as tuplas participantes, a expressão (a) pode conter tuplas de clientes que não tenham um empréstimo na agencia Joinville. Para que o resultado se aproxime do solicitado precisaremos limitar o conjunto de resultados. Será feita uma operação de select no resultado do item (a) buscando apenas aquelas tuplas onde devedor.numero_empréstimo = empréstimo.numero_empréstimo, isto será representado pela expressão (b).

$\sigma_{\text{devedor.numero\_empréstimo = empréstimo.numero\_emprestimo}}(\sigma_{\text{nome\_agencia = "Joinville"}}$ **(devedor x emprestimo)**) (b)

Aplicando sobre a expressão (b) uma operação de projeção buscando o atributo nome_cliente teremos, agora sim, o resultado pretendido.

$\pi_{nome\_cliente}(\sigma_{devedor.numero\_empréstimo = empréstimo.numero\_emprestimo}(\sigma_{nome\_agencia = \text{“Joinville”}}$ **(devedor x emprestimo**))) ( c )

### 6.9.7 Interseção ( ∩ )

Da mesma forma que o operador de união, o operador de interseção exige que seus operandos sejam do mesmo tipo. Dadas duas relações A e B do mesmo tipo, a operação de interseção sobre essas duas relações, é uma relação do mesmo tipo, consistindo em todas as tuplas que pertencem tanto a relação A quanto a relação B.

Exemplo : Encontrar todos os clientes que tenham tanto empréstimo quanto conta.

$\pi_{nome\_cliente}(devedor) \cap \pi_{nome\_cliente}(depositante)$

A operação de interseção pode ser expressa na álgebra relacional em termos de operações de diferença : **r ∩ s = r – ( r – s ).**

### 6.9.8 Join

A operação join (junção) é uma derivação do produto cartesiano. Existem várias formas de join , a mais comum delas é a θ - join , normalmente chamada de join. A operação θ - join de duas relações R e S é denotada por :

$$R \mid><\mid_F S$$

Onde F é a fórmula que especifica o predicado da junção. A join de duas relações é equivalente a construir uma seleção, usando o predicado da join como a fórmula de seleção, sobre um produto cartesiano de duas relações. Desta forma podemos dizer que:

$$R \mid><\mid_F S = \sigma_F (\mathbf{R \times S})$$

Exemplo : Relação de todos os clientes que são depositantes e também devedores.

$depositante \mid><\mid_{depositante.nome\_cliente = devedor.nome\_cliente} devedor$

ou

$\sigma_{depositante.nome\_cliente = devedor.nome\_cliente}$ **(depositante x devedor**)

O exemplo anterior demonstra um caso especial de join chamado de equi-join. Isto acontece quando a fórmula F contém somente operadores aritméticos de igualdade ( = ). Quando

a operação de join acontece sobre um atributo específico comum às duas relações, e não sobre uma fórmula, este tipo de join será chamado de **natural join.** A especificação deste tipo de operação segue ao mesmo formato de join , exceto pelo fato de que não haverá referência a uma fórmula, e sim a um atributo específico.

Notação de **natural join** : $R \mid\!><\!\mid_A S$ , onde A representa um atributo comum para as duas relações R e S.

O conceito de **semijoin** deriva de join. A **semijoin** da relação R , definida sobre o conjunto de atributos A , pela relação S, definida sobre um conjunto de atributos B, é o subconjunto das tuplas de R que participam da join de R com S.

A **semijoin** é denotada por $R \mid\!><_A S$ .

Em uma linguagem mais simples: a semijoin denotada por $R \mid\!><_A S$ irá apresentar como resultado as tuplas de R que contribuíram para formar uma operação de join com a relação S, sobre o atributo A.

# 7. Linguagem SQL

### 7.1 CREATE

### 7.2 CONSTRAINTS

### 7.3 SELECT

É o operador mais básico da linguagem, ele é usado para indicar os atributos que queremos mostrar em nossa consulta. Podemos usá-lo informando campo a campo o resultado que queremos ou então usar o coringa * que é usado para representar todos os registros de uma tabela. No exemplo abaixo estamos trazendo todos os registros da tabela agencia.

```
SELECT * FROM Agencia

SELECT Nome_agencia
      ,Cidade_agencia
      ,fundos
  FROM Agencia
```

### 7.4 FROM

Também faz parte dos operadores básicos da linhagem. É responsável por informar as entidades que estarão participando do retorno dos registros. Nessa clausula, devemos informar todas as entidades que possuem os campos que desejamos mostrar na clausula SELECT. No exemplo anterior usamos a tabela agencia na clausula FROM.

```
SELECT * FROM Conta
```

### 7.5 WHERE

É o operador de filtragem ou condição, ou seja, todas as restrições necessárias são informadas nessa clausula. No exemplo abaixo procuramos os registros da tabela conta que o saldo seja igual a 500.

```
SELECT Nome_agencia
      ,Numero_conta
      ,saldo
  FROM Conta
  WHERE saldo = 500
```

### 7.6 OPERADORES CONDICIONAIS

Estão associados ao operador WHERE, pois eles irão definir os filtros dos conjuntos de resultados que estamos esperando. Abaixo, alguns operadores condicionais mais comuns, = (igual), <> (diferente), >, <, <=, >=, OR (ou), AND (e). No exemplo abaixo combinamos dois operadores condicionais o > (maior) e o AND (e) para que a consulta retorne apenas os registros com saldo maior que 500 e que o nome da agência seja Joinville.

```
SELECT  Nome_agencia ,
        Numero_conta ,
        saldo
FROM    Conta
WHERE   saldo > 500 AND Nome_agencia = 'Joinville'
```

### 7.7 RELACIONAMENTO ENTRE TABELAS

Nesse momento vamos usar o operador WHERE e a condição AND para efetuar os relacionamentos entre as tabelas. Futuramente vamos aprender a maneira mais adequada de efetuar relacionamentos, as chamadas JOINS. No exemplo abaixo, relacionamos a tabela conta coma tabela depositante através da condição WHERE onde seja respeitada a restrição de integridade referencial, ou seja, deve existir um número de conta válido na tabela conta e outro na tabela depositante.

```
SELECT
Nome_agencia,Conta.Numero_conta,saldo,
Nome_cliente,Depositante.Numero_conta
FROM Conta, Depositante
WHERE Conta.Numero_conta=Depositante.Numero_conta
```

### 7.8 ORDER BY

É o operador de ordenação, responsável por ordenar os registros de uma entidade de acordo com a necessidade do usuário. Por padrão ele opera em ASC (Ascendente). No caso de desejar ordenar por ordem decrescente usamos o operador DESC. Os dois operadores podem ser usados juntos, desde que cada um seja usado em atributos diferentes. Utilizando

a mesma consulta do exemplo anterior vamos incluir o comando de ordenação no campo nome da agência. Em seguida ordenamos pelo saldo trazendo os maiores primeiro.

```
SELECT
Nome_agencia,Conta.Numero_conta,saldo,
Nome_cliente,Depositante.Numero_conta
FROM Conta, Depositante
WHERE Conta.Numero_conta=Depositante.Numero_conta
ORDER BY Nome_agencia

SELECT
Nome_agencia,Conta.Numero_conta,saldo,
Nome_cliente,Depositante.Numero_conta
FROM Conta, Depositante
WHERE Conta.Numero_conta=Depositante.Numero_conta
ORDER BY saldo DESC
```

### 7.9 TOP

Operador para restringir o conjunto de resultados retornados. Ele é geralmente usado em conjunto com um número, que define a quantidade de linhas retornadas. É comum usar ele com o operador ORDER BY. Neste exemplo, ainda usando a consulta anterior trazemos apenas o cliente com o maior salvo da agência.

```
SELECT TOP 1
Nome_agencia,Conta.Numero_conta,saldo,
Nome_cliente,Depositante.Numero_conta
FROM Conta, Depositante
WHERE Conta.Numero_conta=Depositante.Numero_conta
ORDER BY saldo DESC
```

### 7.10 LIKE e NOT LIKE

Operador para trabalhar com busca de textos. Geralmente utilizamos ele quando queremos encontrar ou ignorar registros que tenham parte da cadeia de caracteres informado. Usamos o LIKE em conjunto com um operador de % que é conhecido como coringa.

```
SELECT
Nome_agencia,C.Numero_conta,saldo AS [Total em Conta],
Nome_cliente,D.Numero_conta AS 'Conta do Cliente'
FROM Conta AS C, Depositante AS D
WHERE C.Numero_conta=D.Numero_conta
AND Nome_cliente LIKE 'R%'
AND Nome_cliente NOT LIKE 'Laura%'
ORDER BY saldo DESC
```

### 7.11 IN e NOT IN

Muito utilizado para criar subconjuntos de resultados que devem ser incluídos ou ignorados em uma consulta. Ao invés de usar um operador OR ou AND, podemos usar o IN para indicar quais itens serão retornados.

```
SELECT
Nome_agencia,Conta.Numero_conta,saldo,
Nome_cliente,Depositante.Numero_conta
FROM Conta, Depositante
WHERE Conta.Numero_conta=Depositante.Numero_conta
AND Nome_cliente IN ('Rodrigo','Laura')
ORDER BY saldo DESC
```

### 7.12 AS (Alias)

Esse operador ajuda descrição de campos ou tabelas para que se tornem mais amigáveis ao usuário.

```
SELECT
Nome_agencia,C.Numero_conta,saldo AS [Total em Conta],
Nome_cliente,D.Numero_conta AS 'Conta do Cliente'
FROM Conta AS C, Depositante AS D
WHERE C.Numero_conta=D.Numero_conta
AND Nome_cliente IN ('Rodrigo','Laura')
ORDER BY saldo DESC
```

### 7.13 CASE

Operador usado para tratamento de registros de forma condicional. Usamos quando queremos fazer determinadas transformações nos registros que estão sendo retornados em uma consulta

```
SELECT
Nome_agencia,C.Numero_conta,saldo AS [Total em Conta],
Nome_cliente,D.Numero_conta AS 'Conta do Cliente',

CASE
 WHEN saldo >= 1000 THEN 'Select'
 WHEN saldo > 500 AND saldo < 1000 THEN 'Premier'
ELSE 'Standard'
END AS 'Classificação'

FROM Conta AS C, Depositante AS D
WHERE C.Numero_conta=D.Numero_conta
ORDER BY saldo DESC
```

### 7.14 DISTINCT

Esse operador ajuda a eliminar registros duplicados. Em consultas onde queremos uma lista de itens, cidades ou nomes e não queremos repetições de itens usamos o distinct para eliminar esses registros duplicados.

```
SELECT DISTINCT Cidade_agencia
FROM Agencia
```

### 7.15 SUB CONSULTAS OU SUB SELECT

As sub consultas são uma ferramenta útil quando queremos usar um conjunto de resultados para passar como filtro em uma cláusula WHERE. O propósito é usar outra consulta que traga os dados que queremos como filtro e usá-la junto com o comando IN ou NOT IN para filtrar os resultados. No exemplo abaixo queremos mostrar todos os clientes que são depositantes, portanto podemos usar a sub consulta para auxiliar a filtragem.

```
SELECT  Nome_cliente
FROM    Cliente
WHERE   Nome_cliente IN ( SELECT    Nome_cliente
                          FROM      dbo.Depositante )
```

### 7.16 FUNÇÕES DE AGREGAÇÃO

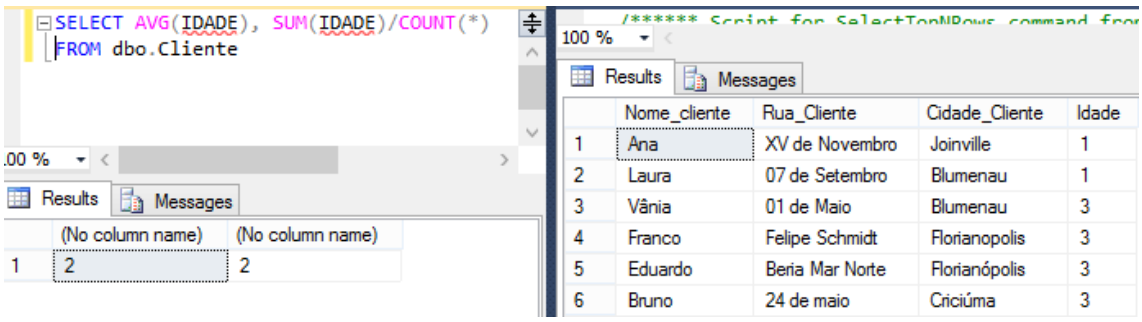
MAX retorna o valor máximo de uma coluna.

```
SELECT MAX ([Total])
  FROM [Bancos].[dbo].[Emprestimo]
```

MIN retorna o valor mínimo de uma coluna

```
SELECT MIN ([Total])
  FROM [Bancos].[dbo].[Emprestimo]
```

AVG retorna o valor médio de uma determinada coluna baseado na soma dos valores dividido pela quantidade de registros correspondentes.




COUNT é utilizada para contar os registros em uma tabela. Podemos ter algumas variações desse comando. Podemos querer contar um campo específico, a quantidade de linhas da tabela, e até registros específicos sem repetição.

```
SELECT COUNT(Numero_emprestimo), COUNT(*), COUNT(DISTINCT Numero_emprestimo)
  FROM [Bancos].[dbo].[Emprestimo]
```

SUM traz o somatório de valores de uma coluna. Nesse caso temos a soma do total de empréstimo da tabela Empréstimo.

```
SELECT sum(Total)
  FROM [Bancos].[dbo].[Emprestimo]
```

GROUP BY É usado para fazer o agrupamento dos demais campos em relação ao campo que você está aplicando a função de agregação, ou seja, se você quiser contar a quantidade de cliente por cidade, quer dizer que você vai contar o nome do cliente e agrupar por cidade. Agora se você quiser apenas o contar os clientes, não precisa usar o group by. Portanto, se usar função de agregação e mais alguma coluna para descrever o agrupamento, então você deve usar o group by.

```
SELECT COUNT(Nome_cliente)
       FROM dbo.Cliente

SELECT Cliente.Cidade_Cliente,
       COUNT(Nome_cliente) FROM dbo.Cliente
              GROUP BY Cidade_Cliente
```

HAVING é utilizado para fazer condições com o resultado das funções de agregação. Como o resultado da função de agregação é feito em tempo de execução do comando, você não pode usar a cláusula WHERE para aplicar uma condição de filtragem. Quando você quiser fazer uma comparação, como por exemplo as agências que tem um total maior que de 2 clientes você deverá usar o having.

```
SELECT Cliente.Cidade_Cliente,  COUNT(Nome_cliente)
       FROM dbo.Cliente
       GROUP BY Cidade_Cliente
              HAVING COUNT(Nome_cliente) > 2
```

### 7.17 JOINS

INNER JOIN é utilizada quando queremos fazer relacionamento entre tabelas. Sua principal característica é ser restritivo, ou seja, só serão retornados os valores que respeitarem as condições da cláusula ON. No exemplo abaixo queremos apenas os clientes que que são devedores.

```
SELECT Cliente.Nome_cliente, devedor.Numero_emprestimo
       FROM dbo.Cliente INNER JOIN dbo.Devedor
              ON (Cliente.Nome_cliente = Devedor.Nome_cliente)
```

LEFT OUTER JOIN é utilizada quando queremos todos os registros da tabela da esquerda, ou seja, a primeira da relação FROM e todos os registros que existam na primeira tabela, mas não obrigatoriamente tenham correspondência na segunda.

```
SELECT Cliente.Nome_cliente, devedor.Numero_emprestimo
FROM dbo.Cliente LEFT OUTER JOIN dbo.Devedor
ON Cliente.Nome_cliente = Devedor.Nome_cliente
```

RIGHT JOIN é parecido com o LEFT, porém nesse caso a tabela mestre da relação é a tabela da direita, ouse seja, serão retornados todos os registros da direita com ou sem relação com a tabela da esquerda.

```
SELECT Conta.*, Agencia.*
       FROM Conta RIGHT JOIN Agencia
              ON (Conta.Nome_agencia = Agencia.Nome_agencia)
```

FULL OUTER JOIN traz os registos das tabelas envolvidas que respeitem ou não as condições, tanto da esquerda quando da direita, além do INNER JOIN normal. Seria uma união de LEFT com RIGHT.

```
SELECT Cliente.Nome_cliente, devedor.Numero_emprestimo
       FROM dbo.Cliente FULL OUTER JOIN dbo.Devedor
              ON Cliente.Nome_cliente = Devedor.Nome_cliente
```

CROSS JOIN executa um produto cartesiano entre as tabelas envolvidas. Se a tabela A tem 3 linhas e a tabela B tem 3, o resultado será uma conjunto de 9 linhas.

```
SELECT Cliente.Nome_cliente, devedor.Numero_emprestimo
       FROM dbo.Cliente CROSS JOIN dbo.Devedor
```

SELF JOIN é utilizada quando queremos relacionar a tabela com ela mesma. Não existe comando SELF JOIN, mas sim a utilização de um Alias com a própria tabela.

```
SELECT T1.[Numero_conta], T2.[Numero_conta]
      FROM Conta T1 INNER JOIN Conta T2
         ON T1.[Numero_conta] = T2.[Numero_conta]
```

### 7.18 UNION e UNION ALL

O operador union server para juntar duas tabelas. Sua função é fazer uma adição dos registros da tabela A com a tabela B. A única condição a se considerar é que os campos que devem ter o mesmo tipo de dados. O union elimina os registros duplicados, já o union All traz todos os registros de ambas as tabelas.

```
SELECT Nome_cliente FROM dbo.Cliente
UNION
SELECT Nome_cliente FROM dbo.Devedor

SELECT Nome_cliente FROM dbo.Cliente
UNION ALL
SELECT Nome_cliente FROM dbo.Devedor
```

### 7.19 DATA E HORA

No SQL Server, a principal função de hora do sistema é o GETDATE(). Com ele conseguimos retornar a data e hora do sistema. A partir dele podemos fazer operações de comparação com a data atual. Também podemos extrair partes da data para auxiliar em comparações. A função DATEPART é usada para retornar uma parte do campo data. Ela precisa sempre da especificação da parte que você quer que seja extraída.

```
SELECT GETDATE()

SELECT MONTH(GETDATE())

SELECT DAY(GETDATE())

SELECT YEAR(GETDATE())

SELECT DATEPART(MINUTE,(GETDATE()))

SELECT DATEPART(HOUR,(GETDATE()))
```

```
SELECT DATEPART(SECOND,(GETDATE()))
```

### 7.20 DATA E HORA

Essa funções podem ser combinadas com campos de data existentes em suas tabelas. No script abaixo é criada uma tabela DATAHORA com uma coluna DATAHORA de onde queremos extrair fragmentos de data. Um dos fragmentos mais comuns é nome do mês. Se a sua versão do SQL Server for diferente do português é necessário informar qual o idioma que você deseja utilizar.

```
CREATE TABLE DATAHORA
(
DATAHORA DATETIME
)
INSERT DATAHORA VALUES ('2014-08-08 19:00:00:000')

SET LANGUAGE PORTUGUESE

SELECT DATENAME(month,(DATAHORA)) FROM DATAHORA

SELECT YEAR(DATAHORA) FROM DATAHORA
```

### 7.21 UPDATE

O comando update server para efetuarmos alterações em registros existentes em uma tabela. Podemos efetuar alterações em uma coluna ou várias, em uma linha ou em um conjunto delas. O comando update só pode ser usado isoladamente em uma tabela, porém a condição pode abranger várias tabelas através de junções entre as tabelas.

```
update Agencia
     set
          Nome_agencia = 'Verde Vale Verde',
          Cidade_agencia = 'Blumenal'
     where Nome_agencia = 'Verde Vale'
```

No exemplo abaixo podemos usar uma junção para validar o update, seguido de uma condição where.

```
update Conta
     set Saldo_Conta = 0
     from Conta Inner Join Emprestimo
on Conta.Codigo_Cliente=Emprestimo.Codigo_Cliente
     Where Emprestimo.Total_Emprestimo < 0
```

### 7.22 INSERT

O comando insert é usado para inserir (criar) novos registros em uma tabela. Podemos efetuar uma inserção de dada vez ou até operações em massa. Para efetuar a inserção você deve indicar quais os campos que você está efetuando a inserção. Em uma tabela que possua uma coluna com um campo que seja do tipo auto incremento, você não precisa especificá-lo na instrução de insert.

```
INSERT CLIENTE
(Codigo_Cliente,Nome_Cliente
,Rua_Cliente,Cidade_Cliente
,Data_Nascimento)
VALUES
(12,'Pedro','João Colin','Joinville','1982-07-08')
```

### 7.23 DELETE

O comando delete é usado para apagar registros de uma tabela.  Também só afeta uma tabela como no update, porém sua condição pode alcançar mais de uma tabela através de junções.

```
DELETE Cliente
     FROM Cliente
     INNER JOIN Emprestimo
on Cliente.Codigo_Cliente=Emprestimo.Codigo_Cliente
     where Emprestimo.Total_Emprestimo < 0
```

### 7.24 VIEWS

As views são usadas para customizar visualizações dos dados ou até abstrair as lógicas implementadas por Joins e outros operadores.

```
CREATE VIEW vSALDO AS SELECT SALDO_CONTA FROM dbo.Conta

SELECT * FROM vSALDO
```

### 7.25 VARIÁVEIS

São utilizadas para auxiliar nas operações de consultas, procedures e funções. Elas trazem dinamismo para as consultas estáticas. Você sempre irá declarar a variável antes do uso e em seguida crie o habito de inicializa-la.

```
DECLARE @NUMERO_CONTA VARCHAR(10)

SET @NUMERO_CONTA = 'C-100'
```

### 7.26 PROCEDURE

As procedures são uma espécie de encapsulamento do código SQL, ou seja, você cria uma série de procedimentos e comandos e os "esconde" dentro da procedure. A procedure pode requerer ou não parâmetros de entrada, que são tratados como variáveis de entrada. Você pode usá-las dentro do código da procedure. Dentro da procedure você também pode declarar novas variáveis, bem como usar diversos outros comandos, como insert, update, delete, ler de views e assim por diante.

```
CREATE PROCEDURE RETORNA_SALDO
(
     @NUMERO_CONTA VARCHAR(10)

)
     AS
     BEGIN
          SELECT SALDO_CONTA FROM Conta
          WHERE NUMERO_CONTA=@NUMERO_CONTA
     END
```

### 7.27 IF

O IF é a estrutura condicional mais simples de todas, onde podemos usá-la isoladamente ou em conjunto com o ELSE. Não esqueça sempre de abrir um BEGIN e fechar com um END cada bloco condicional para não correr o risco da condição entrar no bloco errado.

```
DECLARE @NUMERO_CONTA VARCHAR(10)

SET @NUMERO_CONTA = 'C-100'

IF @NUMERO_CONTA = 'C100'
BEGIN
SELECT 'TRUE'
END
ELSE
BEGIN
```

```
SELECT 'FALSE'
END
```

### 7.28 WHILE

O WHILE é uma estrutura de repetição que podemos usar para criar laços no dentro do código SQL. No SQL não temos a estrutura de repetição FOR.

```
DECLARE @CONTADOR INT
SET @CONTADOR = 1

WHILE @CONTADOR <= 10

BEGIN
    PRINT @CONTADOR
    SET @CONTADOR = @CONTADOR + 1
END
```

### 7.29 TRIGGERS

As triggers ou gatilhos são um importante recurso para os bancos de dados no sentido que possibilitar que eventos sejam disparados cada vez que um registro é inserido, deletado ou atualizado. Suponha que você tem duas tabelas:

```
SELECT * FROM dbo.Conta
SELECT * FROM dbo.Depositante
```

Nosso objetivo é atualizar o saldo da conta automaticamente via banco quando um novo depósito for efetuado.

O Exemplo abaixo mostra como criar uma trigger que será disparada cada vez que um insert ocorrer na tabela depositante.

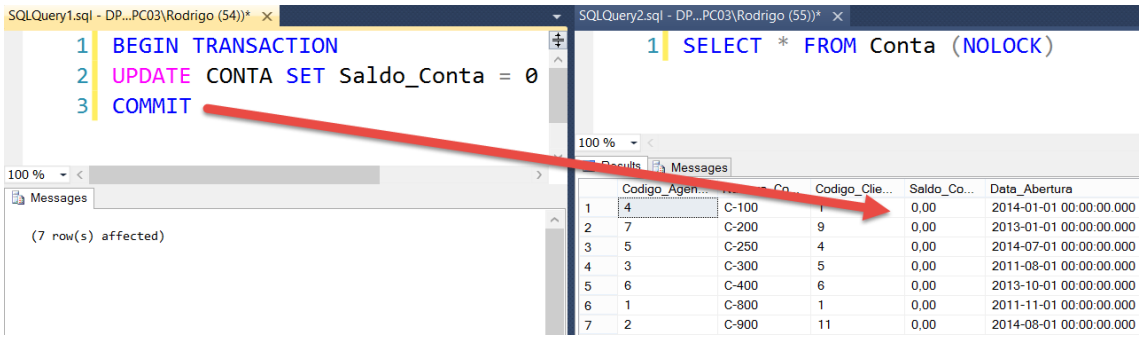
```
CREATE TRIGGER TG_ATUALIZA_SALDO
ON Depositante
FOR INSERT
AS
BEGIN
```

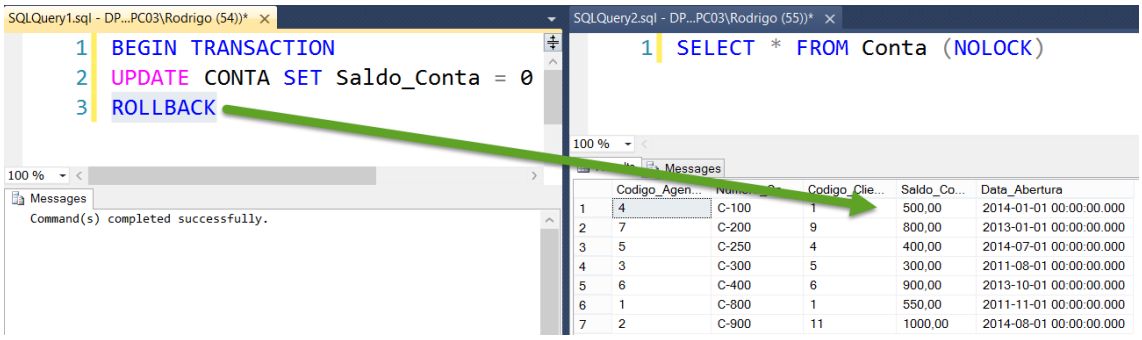
```
UPDATE dbo.Conta SET Saldo_Conta = Saldo_Conta +
(SELECT INSERTED.Valor FROM INSERTED)
WHERE Numero_Conta = (SELECT INSERTED.Numero_Conta
FROM INSERTED)

END
```

### 7.30 TRANSAÇÕES

A estrutura básica de uma transação está condicionada a três fatores. BEGIN TRANSACTION, indica que uma transação foi aberta e que ela deverá ter um COMMIT que irá finalizá-la com sucesso ou então teremos um ROOL BACK que irá desfazê-la, ou seja, vai devolver os dados ao estado original.




```
SELECT * FROM dbo.Conta WHERE Numero_Conta = 'C-100'
```

```
BEGIN TRANSACTION

UPDATE dbo.Conta SET Saldo_Conta = 25000
```

```
WHERE Numero_Conta = 'C-100'

--COMMIT

--ROLLBACK


SELECT * FROM dbo.Conta WHERE Numero_Conta = 'C-100'

SELECT * FROM dbo.Conta WHERE Numero_Conta = 'C-100'
```

### 7.31 FUNÇÕES

As funções são parecidas com procedures, porém são usadas em conjunto com o comando SELECT.

```
CREATE FUNCTION

fn_conver_saldo(@saldo money)

RETURNS money

BEGIN

DECLARE @TOTAL MONEY

SET @TOTAL = @saldo/2.4

RETURN @TOTAL

END

SELECT dbo.fn_conver_saldo(dbo.Conta.Saldo_Conta) FROM dbo.Conta
```

### 7.32 ÍNDICES

Os índices são utilizados para melhorar o desempenho das consultas. Você deve levar em consideração os campos que você está listando no SELECT, os campos usados no WHERE e no ORDER BY.

Repare a consulta abaixo:

```
SELECT DISTINCT dbo.Cliente.Nome_Cliente, dbo.Cliente.Cidade_Cliente

FROM dbo.Cliente

WHERE Nome_Cliente = 'Vânia'

ORDER BY Nome_Cliente DESC
```

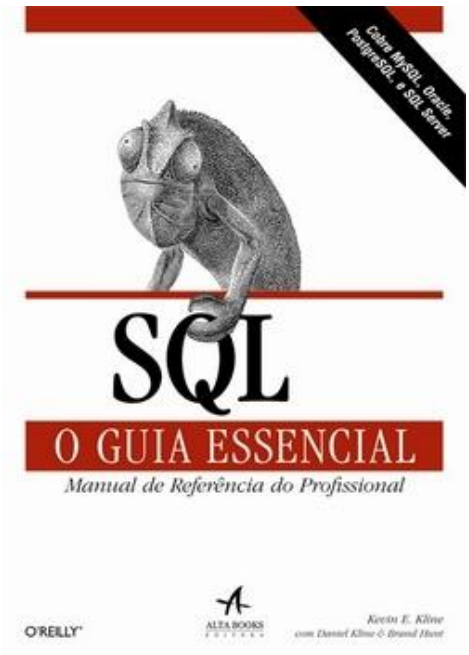
Para otimizar essa consulta podemos criar um índice com os campos Nome_Cliente e Cidade_Cliente, pois estamos querendo exibir o nome e a cidade do cliente. Para otimizar ainda mais, vamos colocar o nome como primeiro campo do índice e ordenamos ele de forma descendente.

Dessa forma ao procurar os registros vai ficar mais rápido, pois ao percorrer o índice ele vai achar o registro Vânia e a cidade dela vai estar ao lado.

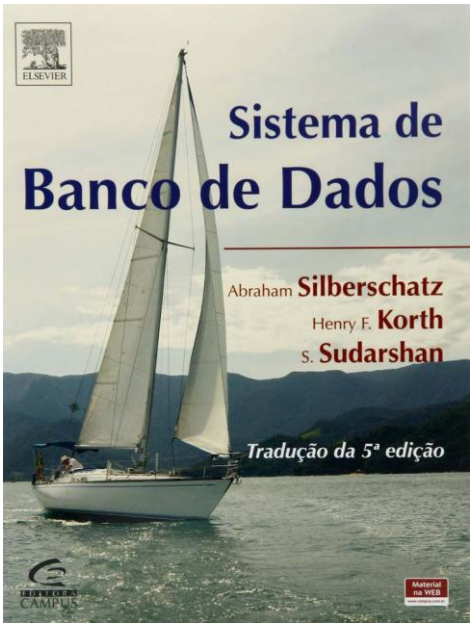
```
CREATE INDEX IX_NomeCidade ON dbo.Cliente

(Nome_Cliente DESC, Cidade_Cliente)
```

# 8 REFERÊNCIAS





# 9 Links

Joins: http://www.codeproject.com/Tips/712941/Types-of-Join-in-SQL-Server

Datas: http://msdn.microsoft.com/pt-br/library/ms174420.aspx

http://msdn.microsoft.com/pt-br/library/ms189491.aspx

http://www.devmedia.com.br/funcoes-de-data-no-sql-server/1946