Banco de Dados

Fernando José Braz

Esta apostila tem como objetivo principal auxiliar os alunos do curso de Sistemas de Informação da Universidade da Região de Joinville – UNIVILLE, no decorrer da disciplina de Banco de Dados. Os assuntos aqui abordados foram pesquisados em diversas fontes bibliográficas que se encontram relacionadas no final deste trabalho. Este trabalho visa facilitar a iniciação do estudante na área de Banco de Dados, oferecendo um requisito mínimo para que o aluno possa aprofundar-se na literatura especializada.

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.

Os sistemas de banco de dados devem garantir o armazenamento correto, seguro e eficiente dos dados. Efetuando o armazenamento sob esta forma fica praticamente garantida a otimização e a segurança da recuperação das informações, fator determinante do sucesso de uma aplicação de armazenamento de informações.

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 e o modelo orientado a objetos. O foco deste trabalho é apresentar o conceito de banco de dados de acordo com o modelo relacional, neste momento apenas será feita uma breve apresentação de cada modelo.

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

3.3.1 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

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.

3.3.2 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 ---------

Freqüentemente 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.

4 Arquitetura de Sistemas de Bancos 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 conseqüência 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 independencia 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 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).

5.1 Conjunto de Entidades

No Modelo Entidade-Relacionamento, as **entidades** irão identificar e representar os objetos do mundo real, podem tanto ser concretas 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 conseqüência, 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 !!!).

5.2 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.**

5.3 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.

5.4 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.

5.5 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.

5.5.1 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.

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.

5.5.2 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**.**

5.5.3 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.

5.6 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.

5.7 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.**

5.7.1 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.

5.7.2 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.

5.7.3 Herança de Atributos

A herança de atributos é uma conseqüência 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.

5.8 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*.

5.8.1 Operações fundamentais

As operações *select, project* e *rename* são classificadas como operações primárias, pois trabalham apenas sobre uma única relação. As operações que trabalham sobre pares de relações são definidas como operações binárias.

5.8.1.1 Select (σ)

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 σ. A seguir alguns exemplos do operador *select.*

Selecionar todas as tuplas da relação *agencia* cuja agência esteja localizada na cidade de "Joinville" : $\sigma_{\text{cidade\_agencia} = \text{"Joinville"}}(\text{agencia})$

Selecionar todas as tuplas da relação *empréstimo* cujos totais são superiores a 2300 :

$\sigma_{\text{total} > 2300}(\text{emprestimo})$

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

Selecionar todas as tuplas da relação *empréstimo* efetuadas na agencia "Blumenau" e cujos totais sejam maiores que 1500 : $\sigma_{\text{nome\_agencia} = \text{"Blumenau"} \wedge \text{total} > 1500}(\text{emprestimo})$

Selecionar todas as tuplas da relação *agencia* para as agências localizadas na cidade de Florianópolis e que possuam fundos maiores ou iguais a 8000 : $\sigma_{\text{cidade\_agencia} = \text{"Florianopolis"} \wedge \text{fundos} \geq 8000}(\text{agencia})$.

5.8.1.2 Project (π)

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 π, 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*: $\pi_{\text{nome\_agencia}}(\text{conta})$;

Listar os nomes de todos os clientes da relação *cliente*: $\pi_{\text{nome\_cliente}}(\text{cliente})$;

Listar os nomes dos clientes e rua da relação *cliente*: $\pi_{\text{nome\_cliente, rua\_cliente}}(\text{cliente})$.

Poderíamos aumentar um pouco a complexidade das consultas se solicitarmos por exemplo: ***" o nome dos clientes que moram em Florianópolis "*** . Para retornar o conjunto de resultados desta consulta iremos combinar duas operações da álgebra relacional : a ***select*** e a ***project*** . Estaremos portanto, trabalhando com uma expressão em álgebra relacional. A sintaxe da expressão solicitada seria a seguinte:

$$\pi_{\text{nome\_cliente}}(\sigma_{\text{cidade\_cliente} = \text{"Florianópolis"}}(\text{cliente}))$$

Para construir a expressão, em vez de dar o nome da relação como argumento na operação de *projeção* (π), estamos informando uma expressão (σ) que traz como resultado uma relação.

5.8.1.3 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.*

Tomando como base as relações apresentadas até aqui, fica claro supor que não existe a possibilidade de executar-se uma operação *union* entre as relações *conta* e *devedor* , pois esta operação viola a condição do item (a). Da mesma forma, uma operação de *union* entre o conjunto de nomes de clientes e o conjunto de cidades, não seria possível pois estaria violando a restrição do item (b).

Uma consulta possível de ser executada com a operação de *union* seria aquela que solicita o nome de todos os clientes do banco que tenham uma conta, um empréstimo ou ambos. Para construir esta consulta, iremos operar em três fases distintas:

1) consulta que retorna os nomes dos clientes devedores: $\pi_{\text{nome\_cliente}}$ (devedor)
2) consulta que retorna os nomes dos clientes com conta: $\pi_{\text{nome\_cliente}}$ (depositante)
3) operação *union* entre os dois conjuntos anteriores: $\pi_{\text{nome\_cliente}}$ (devedor) $\cup$ $\pi_{\text{nome\_cliente}}$ (depositante)

5.8.1.4 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: πnome_cliente(depositante) - πnome_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.*

5.8.1.5 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 $P = R \times S$ 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.

$\sigma_{\text{nome\_agencia} = \text{“Joinville”}}$ **(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.

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

5.8.1.6 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.

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

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

5.8.1.7 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 :

$$R \ |><|_F \ S$$

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:

$$R \ |><|_F \ S \ = \sigma_F \ \mathbf{(R \ x \ S)}$$

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

$$depositante \ |><|_{depositante.nome\_cliente = devedor.nome\_cliente} \ devedor$$

ou

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

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.

5.9 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*[1] (*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*)

[1] *OLTP*: ambientes OnLine Transaction Processing são baseados em transações intensivas, alterações das informações são construídas através do uso de transações com o banco de dados.

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*[2] (*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.

PRIMEIRA FORMA NORMAL - 1FN

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,

[2] *OLAP*: ambientes OnLine Analytical Processing são baseados em resumos de dados, servindo como base para a análise de tendências e comportamento dos ambientes comerciais.

para uma única ocorrência de valor da chave deve existir uma única ocorrência dos demais atributos.

SEGUNDA FORMA NORMAL – 2FN

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).

TERCEIRA FORMA NORMAL – 3FN

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.

Na seqüência deste material seguem dois exemplos de normalização de dados. O primeiro deles diz respeito às informações contidas em uma nota fiscal, o segundo exemplo repete o processo de normalização para o cadastro de um funcionário. Vale lembrar que os exemplos são de caráter didático, desta forma a busca de entidades e atributos nos dois exemplos não abrange todas as possibilidades de uma modelagem para um ambiente de dados real.

Normalização NOTA FISCAL

Relação original

Notas_fiscais (***num_NF***, serie, dt_emissão, cod_cliente, nome_cliente, end_cliente, cgc_cliente, cod_mercad, desc_mercad, qtd_vendida, preco_venda, tot_venda_mercad, tot_nota)

1FN

Notas_fiscais (***num_NF***, serie, dt_emissão, cod_cliente, nome_cliente, end_cliente, cgc_cliente, tot_nota)

Vendas (***num_NF***, ***cod_mercad***, desc_mercad, qtd_vendida, preco_venda, tot_venda_mercad)

2FN

Notas_fiscais (***num_NF***, serie, dt_emissão, cod_cliente, nome_cliente, end_cliente, cgc_cliente, tot_nota)

Vendas (***num_NF***, ***cod_mercad***, qtd_vendida, tot_venda_mercad)

Mercadorias (***cod_mercad***, desc_mercad, preco_venda)

3FN

Notas_fiscais (***num_NF***, serie, dt_emissão, cod_cliente, tot_nota)

Vendas (***num_NF***, ***cod_mercad***, qtd_vendida, tot_venda_mercad)

Mercadorias (***cod_mercad***, desc_mercad, preco_venda)

Clientes (***cod_cliente***, nome_cliente, end_cliente, cgc_cliente)

Normalização FUNCIONARIO

Relação original

Funcionario (***mat_func***, nome_func, dt_nasc_func, cpf_func, fone_func, endereco_func,

cod_cargo, desc_cargo, valor_cargo, dt_admissão_func, cod_depto, desc_depto,

cod_curso, desc_curso, dt_curso)

1FN

Funcionario (***mat_func***, nome_func, dt_nasc_func, cpf_func, fone_func, endereço_func, cod_cargo, desc_cargo, valor_cargo, dt_admissão_func, cod_depto, desc_depto)

Curso_funcionário (***mat_func, cod_curso***, desc_curso, dt_curso)

2FN

Funcionario (***mat_func***, nome_func, dt_nasc_func, cpf_func, fone_func, endereço_func, cod_cargo, desc_cargo, valor_cargo, dt_admissão_func, cod_depto, desc_depto)

Curso_funcionário (***mat_func, cod_curso***, dt_curso)

Curso (***cod_curso, desc_curso***)

3FN

Funcionario (***mat_func***, nome_func, dt_nasc_func, cpf_func, fone_func, endereço_func, cod_cargo, dt_admissão_func, cod_depto)

Cargo (***cod_cargo***, desc_cargo, valor_cargo)

Depto (***cod_depto***, desc_depto)

Curso_funcionário (***mat_func, cod_curso***, dt_curso)

Curso (***cod_curso, desc_curso***)

6 Linguagem de quarta geração e *SQL*

A linguagem de quarta geração foi desenvolvida no final dos anos 70 e utilizada em grande escala no desenvolvimento de aplicações. As ferramentas de quarta geração incluem softwares de consulta, geradores de relatórios, softwares de gráficos, geradores de aplicação e outras ferramentas que reduzem drasticamente o tempo de desenvolvimento. As linguagens de quarta geração são do tipo **não-procedurais**, podem ser classificadas em 05 grupos principais: **linguagens de consulta, linguagens gráficas, geradores de relatórios, geradores de aplicações** e **linguagens de programação de altíssimo nível.**

- **Linguagens de consulta (query languages):** são linguagens de alto nível, possibilitam o acesso aos dados mantidos em sistemas de informação. Possibilitam solicitações *ad-hoc* no acesso aos dados, aumentando desta forma a interatividade entre o usuário e o sistema de banco de dados.
- **Linguagens de gráficos:** são softwares especializados em exibir dados em forma gráfica, permitem a visualização de tendências ou comportamentos das informações.
- **Geradores de relatórios:** são ferramentas de software que exibem as informações, ou resumo delas, mantidas em sistemas de banco de dados sob a forma de relatórios. Podem tanto ser *on-line* quanto ter o seu processamento em lotes, dependendo do ambiente em que se encontra inserido.
- **Geradores de aplicações:** são pacotes relacionados de software que podem gerar aplicações completas de sistemas de informação sem programação

personalizada. O usuário final só precisa especificar o que é necessário a ser feito, o gerador de aplicações se encarregará de escrever o programa necessário. Os geradores de aplicações mais poderosos oferecem uma série de recursos integrados como linguagens de consulta, geradores de gráficos e relatórios.

- **Linguagens de altíssimo nível:** são ferramentas para desenvolvedores profissionais, produzem códigos de programas com muito menos instruções.

6.1 SQL – Structured Query Language

A SQL – Structured Query Language – é a linguagem de consulta mais utilizada no mercado, esta linguagem utiliza uma combinação de construtores em álgebra e cálculo relacional. A versão original foi desenvolvida pela IBM no Laboratório de Pesquisa de San José, originalmente a linguagem era chamada de Sequel, evoluindo até que teve o nome alterado para SQL. Em 1986, o American National Standards Institute (ANSI) e a International Standards Organization (ISO) publicaram os padrões para a SQL, chamada SQL-86. Uma extensão para os padrões foi publicada em 1989, a SQL-89. Atualmente, a versão em uso é a SQL-92, no entanto, a padronização da SQL3 já se encontra em andamento.

A linguagem SQL possui várias partes, cada qual composta por uma série de comandos específicos:

- **Linguagem de definição de dados (*data definition language – DDL*):** a *SQL DDL* oferece comandos para a definição de esquemas de relações, exclusão de relações, criação de índices e modificação nos esquemas de relações.
- **Linguagem de manipulação de dados (*data manipulation language – DML*):** a *SQL DML* abrange uma linguagem de consulta baseada tanto na álgebra relacional quanto no cálculo relacional de tuplas. Engloba também comandos para inserção, exclusão e modificação de tuplas no banco de dados.
- **Definição de visões:** pode ser considerada uma parte da DDL, pois se preocupa da criação de visões no banco de dados.
- **Autorização:** a SQL DDL engloba comandos para a atribuição de direitos e permissões de acessos aos objetos do banco de dados.
- **Integridade:** a SQL DDL possui comandos para a definição e imposição das restrições de integridade no banco de dados.
- **Controle de transações:** sob este item estão classificadas as estruturas de controle das transações, a concorrência e a atomicidade das transações são garantidas por este controle.

6.1.1 Estruturas Básicas

A estrutura básica de uma expressão em SQL consiste em 03 cláusulas: **select, from** e **where**, a seguir uma breve descrição de cada uma das cláusulas:

- **Select:** corresponde à operação de **projeção** da álgebra relacional. Utilizada para identificar os atributos desejados para o resultado de uma consulta.
- **From:** corresponde à operação de **produto cartesiano** da álgebra relacional. Associa as relações que serão pesquisadas durante a execução da expressão.
- **Where:** identifica a seleção do predicado da álgebra relacional.

Considerando a seguinte consulta em SQL: **Select** A_1, A_2, A_3 **From** r_1, r_2, r_3 **where** P, a linguagem SQL construirá um produto cartesiano entre as relações constantes da cláusula **From**, executa uma operação de seleção em álgebra relacional usando o predicado da cláusula **where** e, finalmente, projeta o resultado sobre os atributos da cláusula **select**. A bem da verdade, existe um mecanismo de otimização do plano de execução da consulta, visando obter o plano de execução que ofereça o menor custo de processamento para a execução da consulta. Este plano de execução poderá alterar a ordem das operações que deverão ser efetuadas para a execução da expressão da SQL.

- **<u>Cláusula Select</u>**

  O resultado de uma consulta é uma relação, uma relação pode ser considerada um conjunto, desta forma, dentro de uma relação não existirão tuplas duplicadas. A cláusula **select** indica o conjunto de atributos sobre os quais deve ser projetado o resultado de uma operação de seleção. Na cláusula **select** serão indicados os atributos para os quais a resposta da consulta deve ser projetada. Na cláusula, pode ser indicada uma série de atributos, podem ser indicados todos os atributos de uma relação através

da utilização do símbolo *. Ainda na cláusula select é possível envolver operações sobre o conjunto dos atributos. Para obedecer ao preceito da não duplicidade de tuplas em uma relação, seria necessário que os resultados duplicados fossem excluídos, este procedimento representa um custo adicional ao processamento da consulta. Desta forma, a maioria dos bancos de dados permitem que a duplicidade de tuplas aconteça no resultado de uma cláusula **select**, porém oferecem a possibilidade de eliminar a duplicidade através da palavra-chave **distinct**.

Exemplos:

Select cod_cliente, nome_cliente, fone_cliente **from** clientes

Select distinct cod_cliente, nome_cliente, fone_cliente **from** clientes

Select * from clientes

Select nome_cliente, saldo_cliente * 1.10 **from** clientes

- **Cláusula Where**

A cláusula **where** apresenta a seleção do predicado da expressão relacional, é através desta cláusula que o conjunto de resultados do produto cartesiano poderá ser restringido. Os operadores lógicos **and, or** e **not** poderão ser utilizados, bem como os operadores de comparação **=, <, >, <>, =>, <=.**

Exemplos:

Select cod_cliente, nome_cliente **from** clientes **where** cod_cliente >= 100

Select cod_cliente, nome_cliente **from** clientes **where** dt_nasc > ‘10/01/1980’ **and** cod_cliente < 1000.

- **Cláusula From**

A cláusula **from** define um produto cartesiano das relações na cláusula. Supondo que estivéssemos interessados em encontrar todos os nomes dos clientes e seus dependentes, logicamente somente dos clientes que possuem algum dependente. A consulta em SQL poderia ser expressa da seguinte forma:

Select clientes.nome_cliente, dependentes.nome_dependente **from** clientes, dependentes **where** clientes.cod_cliente = dependentes.cod_cliente

Na expressão acima se pode verificar a operação de produto cartesiano entre as duas relações **clientes** e **dependentes,** na cláusula **where** acontece a limitação do conjunto de resultados.

6.2 Ordenação de Tuplas

A cláusula **order by** permite que o resultado de uma consulta seja ordenado segundo algum atributo. Por padrão a cláusula **order by** classifica os resultados em ordem ascendente, para que esta classificação mude para descendente será necessário incluir a palavra-chave **desc** na sintaxe da consulta. A ordenação pode ser realizada com base em um único atributo ou em uma série deles. A consulta que mostra o código do cliente e o seu nome, ordenados de forma descendente com base no nome do cliente, poderia ser expressa

da seguinte forma: **Select** cod_cliente, nome_cliente **from** clientes **order by** nome_cliente **desc.**

6.3 Funções Agregadas

As funções agregadas possuem a capacidade de agregar, resumir um conjunto de valores relativos ao resultado de uma expressão, de forma a apresentar um único valor como resultado final. A SQL possui 05 funções agregadas:

- **Avg -** (average) – Média
- **Min -** (minimum) – Mínimo
- **Max -** (maximum) – Máximo
- **Sum -** (total) – Total
- **Count -** (count) – Contagem

A instrução a seguir apresenta o valor da média dos saldos dos clientes, portanto, apresentará um valor único: **Select avg(saldo_cliente) from clientes.** Esta instrução apresenta como resultado um único atributo e uma única linha de registro. É possível a formação de grupos de tuplas, de forma que cada grupo apresente um valor resumido pela função de agregação. Isto é possível pela utilização da palavra-chave **group by,** o atributo ou atributos da cláusula **group by** irão indicar a formação dos grupos de resumos. Por exemplo, a relação **pedidos** identifica os pedidos emitidos para os clientes da empresa, desta forma posso montar uma consulta que mostre o número de pedidos por código do cliente: **Select cod_cliente, count(cod_pedido) from pedidos group by cod_cliente**. A função **count(*)** não elimina as duplicidades e não ignora os valores nulos.

A cláusula **having** atua nas funções de agregação da mesma forma que a cláusula **where** atua na restrição das tuplas. A cláusula **having** seleciona os grupos após as suas formações, agindo desta forma, sobre os grupos e não sobre as tuplas que são envolvidas naqueles resumos de dados. Por exemplo, podemos mostrar os a quantidade de pedidos por código de cliente, porém somente daqueles clientes que possuem mais do que 05 pedidos:

Select cod_cliente, count(cod_pedido) from clientes group by cod_cliente

Having count(cod_pedido) > 5.

6.4 Visões de Dados

As **visões** trazem a possibilidade de esconder do usuário a complexidade do banco de dados, possibilitam a limitação do acesso aos dados. Na linguagem SQL a criação de visões acontece através da seguinte sintaxe:

Create view nome_da_visão **as** < expressão da consulta >

Create View primeira_visao **as** Select * from clientes where cod_cliente > 10

É importante observar que as visões não armazenam dados, apenas contêm a "receita" de uma consulta a ser executada.

6.5 Exclusão de Tuplas

A operação de exclusão de dados pode ser expressa pela seguinte sintaxe:

Delete from nome_da_relação **where** predicado.

Para a exclusão de todos os registros de clientes cuja data de nascimento seja maior que 01/01/1980 a instrução será a seguinte:

Delete from clientes **where** dt_nascimento > ‘01/01/1980’.

6.6 Inclusão de Tuplas

A inserção de dados em uma relação tanto pode acontecer através da especificação das tuplas uma a uma, quanto pela inclusão do resultado de uma consulta. O comando de inserção deve fornecer todos os atributos necessários para a inclusão correta na relação, os valores a serem inseridos devem pertencer ao mesmo domínio dos atributos da relação onde se inserem os dados.

Inserção individual:

Insert into clientes (cod_cliente, nome_cliente, cpf_cliente)

values (‘1000’, ‘Jose da Silva’, ‘12345678911’)

Inserção em grupo:

Insert into clientes (cod_cliente, nome_cliente, cpf_cliente)

Select cod_cliente, nome_cliente, cpf_cliente **from** clientes_antigos

6.7 Atualização de Tuplas

A operação de atualização permite que o usuário altere valores de alguns atributos da relação, mais que isso, permite que se alterem apenas algumas tuplas específicas.

Update nome_da_relação **Set** atributo = novo_valor **where** Predicado

Atualização do valor do saldo dos clientes com código maior que 500, o saldo sofrerá um acréscimo de 10 %.

Update clientes **Set** saldo = saldo * 1.10 **Where** cod_cliente > 500.

6.8 Operações de Conjuntos

Os operadores **union, intersect** e **except** operam relações e correspondem às operações $\cup$, $\cap$ e $-$ da álgebra relacional. Como a união, interseção e diferença de conjuntos da álgebra relacional, as relações participantes das operações precisam ser compatíveis, ou seja, precisam ter o mesmo conjunto de atributos.

6.8.1 União

O operador **union** apresenta o resultado da união de duas relações, é necessário que as relações possuam o mesmo número de atributos. Na construção da união, as duplicidades serão eliminadas do conjunto de resultados existe, porém, a possibilidade de manutenção das duplicidades, bastando que se utilize a cláusula **all** após a instrução **union**.

Select cod_cliente **from** clientes

Union

Select cod_dependente **from** dependents

6.8.2 Interseção

A operação de interseção irá encontrar todas as tuplas que participam da interseção das relações. Por exemplo, para encontrar todos os clientes que são funcionários, a expressão SQL seria a seguinte:

Select nome_cliente **from** clientes

Intersect

Select nome_funcionario **from** funcionários

6.8.3 Operação Exceto

Este operador encontra todas as tuplas que participam de uma relação, mas não participam de outra. Por exemplo, para saber quais os clientes que não estão na relação funcionários a consulta a ser executada poderia ser a seguinte:

Select nome_cliente **from** clientes

Except

Select nome_funcionário **from** funcionários

6.9 Linguagem de Definição de Dados (DDL)

A linguagem SQL contém uma série de instruções definida para a construção do conjunto de relações, pode também definir o esquema das relações, domínio de cada atributo, regras de integridade, índices etc.

Para a definição dos domínios dos atributos das relações é necessário que se apresente uma relação dos principais domínios:

- **Char(n):** cadeia de caracteres de tamanho fixo de tamanho **n** a ser definido pelo usuário.
- **Varchar(n):** cadeia de caracteres de tamanho variável, com um tamanho máximo de **n** caracteres.
- **Int:** número inteiro válido
- **Smallint:** número inteiro, porém com uma abrangência menor que o tipo **int**
- **Numeric(p,d):** número de ponto fixo, o valor **p** identifica o número de dígitos, o valor **d** informa quantos dígitos estarão à direita do ponto decimal.
- **Real, doublé precision:** número de ponto flutuante com ponto flutuante de precisão dupla, esta precisão é dependente do equipamento.
- **Float(n):** número de ponto flutuante, a precisão **n** é definida pelo usuário.
- **Date:** calendário contendo um ano, mês e dia do ano.
- **Time:** horário em horas, minutos e segundos.

Pode ser necessário proibir a permanência de valores nulos para determinados atributos, para tanto a SQL permite que se especifique a condição **not null** no momento da criação da relação. Caso a lista de domínios não seja suficiente para os objetivos do usuário, é possível a criação de domínios "personalizados" através da instrução **create domain.**

Create domain cpf **char(11)** -- esta instrução cria um domínio chamado **cpf** constituído de 11 caracteres.

6.9.1 Definição de Esquema

Uma relação (tabela) pode ser definida através do comando **create table** que possui a seguinte sintaxe:

Create table nome_da_relação **(** A_1 D_1, A_2 D_2, A_3 D_3, ..., A_n D_n, <regras de integridade$_1$>, <regras de integridade$_n$> **)**

Onde A identifica o nome do atributo e D o domínio daquele atributo. As regras de integridade podem ser a chave primária, as chaves estrangeiras, as restrições do tipo check etc. No momento iremos nos ater à restrição do tipo chave primária (primary key).

O comando **primary key (a1, a2, a3)** diz que a chave primária da relação será composta pelo conjunto de atributos a1, a2 e a3. A seguir apresentaremos um exemplo da instrução de criação de relação.

Create Table funcionarios

(cod_funcionario **char(5) not null**,

nome_funcionario **varchar(30) not null**,

fone_funcionario **varchar(10),**

primary key (cod_funcionario) **)**

A instrução **drop table** exclui a relação do banco de dados, elimina o esquema da relação banco de dados.

7 Sistema de Gerenciamento de Banco de Dados

O Sistema Gerenciador de Banco de Dados (SGBD) é o software que gerencia todo o acesso ao banco de dados. Sempre que o usuário emite uma consulta ao banco de dados, o SGBD intercepta e analisa este pedido. O SGBD, através dos seus componentes, atua nas diversas camadas de acesso aos dados: o nível externo, o mapeamento externo/conceitual, o nível conceitual, o mapeamento conceitual/interno e o nível interno, buscando então as definições de armazenamento dos dados. Este processo de análise e interpretação dos comandos e esquemas de dados tem um alto custo para o SGBD, devido a este fato alguns produtos de SGBD oferecem a possibilidade de utilização de instruções compiladas no momento da sua execução. Esta possibilidade traz uma redução no custo de processamento, visto que o processo de interpretação e análise dos esquemas não é mais necessário, basta que se utilize o plano de execução que já se encontra compilado no SGBD. A seguir apresentaremos as principais funções de um SGBD.

7.1 Definição de Dados

O SGBD deve ter a capacidade de trabalhar com as definições de dados (esquemas externo, conceitual e interno e todos os mapeamentos) em forma fonte e convertê-los para a forma objeto apropriada. Ou seja, o SGBD deve incluir componentes de processador de DDL ou compilador de DDL para cada linguagem de definição de dados.

7.2 Manipulação de Dados

O SGBD deve ser capaz de trabalhar com as requisições do usuário, tais requisições podem ser operações de inserção, atualização e exclusão de dados. Ou seja, o SGBD deve conter um processador ou compilador de DML.

7.3 Otimização e Execução

As requisições de DML devem ser processadas pelo **otimizador**, cuja função é determinar um modo eficiente de implementar a requisição. As requisições otimizadas são executadas sob o controle do **gerenciador** em tempo de execução.

7.4 Segurança e Integridade

A segurança e a integridade dos dados é uma das principais funções do SGBD, a verificação das permissões de acesso e a manutenção das estruturas de controle de integridade serão verificadas pelo SGBD, estas verificações poderão acontecer tanto em tempo de execução quantos em tempo de compilação e análise dos comandos.

7.5 Recuperação e Concorrência

A recuperação de falhas nas transações e o controle da concorrência destas aos dados, é também uma das funções do SGBD. É interessante frisar que o aumento da concorrência, na maior parte das vezes, implica em um relaxamento na integridade das informações recuperadas através das transações.

7.6 Dicionário de Dados

É natural que o SGBD forneça um dicionário de dados, que tem a função de armazenar informações sobre o sistema de banco de dados, os metadados ou "dados sobre os dados" compõem o dicionário de dados.

7.7 Processamento de Consultas

Segundo Molina et al. (2001), existem dois caminhos possíveis para que as ações dos usuários afetem o banco de dados: **Resposta de uma Consulta** e o **Processamento de Transações.**

Resposta de uma Consulta: a consulta é analisada gramaticalmente e aperfeiçoada por um *compilador de consultas*. Estas ações resultam em um *plano de consulta*, que corresponde à seqüência de ações a serem executadas para a resposta da consulta. **O** *plano de consulta* é repassado para o *mecanismo de execução.* O *mecanismo de execução* tem um papel fundamental no processo de execução de uma consulta, este mecanismo emite uma série de requisições de itens de dados (registros ou tuplas) para o gerenciador de recursos. O gerenciador de recursos "conhece" a localização física e lógica dos itens solicitados, as informações são então enviadas do local de armazenamento (em geral discos) para o buffer de memória, possibilitando desta forma uma maior velocidade no acesso aos dados. De posse destes dados, o *mecanismo de execução* tem a possibilidade de processar as operações necessárias e entregar os resultados da consulta solicitada.

Processamento de Transações: as consultas e outras ações são agrupadas em transações, que são unidades que têm de ser executadas obedecendo às propriedades ACID.

O processador de transações é dividido em duas partes principais: ***gerenciador de controle de concorrência*** e o ***gerenciador de registro de log e recuperação.***

7.7.1 Transações

Uma transação é uma coleção de operações nos dados, de maneira a levar o banco de dados de um estado consistente a outro igualmente consistente.

Uma transação mantém o banco de dados em estado íntegro através da observância das propriedades ***ACID***.

Atomicidade

Consistência

Independência

Durabilidade

7.7.2 Gerenciamento das Transações e Controle de Concorrência

Os componentes de gerenciamento de transações tem a responsabilidade da manutenção da integridade do banco de dados, através da aplicação das propriedades ***ACID***. Já o controle de concorrência visa gerenciar o acesso concorrente aos dados por parte das várias transações que podem estar sendo executadas em um mesmo instante de tempo.

Os esquemas de controle de concorrência funcionam basicamente atrasando uma operação ou abortando a transação que emitiu essa operação. O protocolo de bloqueio é um

conjunto de regras que determinam quando uma transação pode bloquear e desbloquear cada ítem de dados. Existem uma série de tipos de bloqueios utilizados pelos variados protocolos, os principais são os seguintes:

Compartilhado (S): este tipo de bloqueio permite que outras transações tenham acesso de **leitura** no ítem de dados, porém impede a **escrita** neste ítem de dados por parte de outras transações.

Exclusivo (X): este tipo de bloqueio impede qualquer tipo de acesso ao ítem de dados por parte de outras transações.

7.8 Recuperação de Falhas

Da mesma forma que outros tipos de sistemas, um sistema de banco de dados está sujeito à ocorrência de falhas que podem comprometer ou danificar totalmente o ambiente de banco de dados. Segundo SILBERSCHATZ et al. (1999), as falhas podem ser classificadas em:

Falhas de Transação

Queda de Sistema

Falha de Disco

O armazenamento dos dados é um fator crucial nos mecanismos de recuperação de falhas, vamos apresentar a seguir os principais tipos de armazenamento e suas características:

Armazenamento Volátil: a informação residente em armazenamento volátil, geralmente não resiste a quedas do sistema. Ex: memória principal e memória cache.

Armazenamento Não-Volátil: a informação residente neste tipo de meio sobrevive a quedas do sistema. Ex: discos e fitas magnéticas.

Armazenamento Estável: este tipo de armazenamento garante que a informação residente "nunca" será perdida. Para que este objetivo seja atingido, algumas técnicas são observadas: *RAID* de discos, replicação de dados, sistemas de bateria.

7.8.1 Arquivo de Log e Registros de Checkpoint

A estrutura mais utilizada para a gravação de modificações no banco de dados é o ***arquivo de log***. O arquivo de log é uma sequência de registros de todas as transações que alteraram o estado do banco de dados. Podemos identificar alguns campos utilizados em um arquivo de log: Identificador de transação, Identificador de item de dados, Valor antigo, Valor novo.

O uso de registros de *checkpoint* no processo de recuperação de falhas de bancos de dados possibilita limitar a pesquisa dos registros no arquivo de *log.* O *checkpoint* pode ser considerado como um limitador, indicando que o banco de dados até aquele ponto está atualizado e consistente (OZSU & VALDURIEZ 1999). O processo de refazer as transações (*log redo*) tem início a partir do *checkpoint,* enquanto que o processo de desfazer (*log undo*) retorna somente até aquele ponto.

A seguir apresentaremos a forma de aplicação dos arquivos de log nos principais SGBD's:

DB2 – IBM

Log circular – reutilizável (falhas do sistema).

Log retain – operações redo e undo.

Oracle

Utiliza um arquivo de *log* do tipo *undo*. Durante a operação de confirmação da transação e antes do banco de dados ser atualizado um *log* (temporário) é criado.

Sybase

Transaction log.

Transaction log mirror.

SQL SERVER – Microsoft

Transaction Log – operações redo e undo.

8 Administração de Banco de Dados

A administração de Banco de Dados é uma tarefa que requer conhecimento tanto do negócio da empresa quanto dos conceitos de bancos de dados. Além disso, é absolutamente essencial que se domine a ferramenta de gerenciamento do banco de dados, este conhecimento poderá otimizar algumas tarefas administrativas e abrir a possibilidade de implementações de rotinas que garantam a disponibilidade do banco de dados para os usuários. A seguir apresentaremos algumas das atividades principais desenvolvidas pelo administrador de banco de dados.

8.1 Definição de Esquema

O administrador do banco de dados irá definir quais informações devem ser mantidas no banco de dados. O DBA cria o esquema do banco de dados escrevendo um conjunto de definições que são transformadas em um conjunto de tabelas armazenadas no dicionário de dados.

8.2 Definição da estrutura de dados e método de acesso.

O DBA cria estruturas de dados e métodos de acesso apropriados escrevendo um conjunto de definições, que são traduzidas pelo compilador da linguagem de definição de dados.

8.3 Ligação com os usuários

É função do DBA promover a ligação dos usuários com o banco de dados, esta ligação acontece garantindo a disponibilidade das informações que os usuários necessitam do banco de dados. O auxílio no desenvolvimento de esquemas externos (visões de dados) é também uma das funções do DBA. A ligação com o usuário também acontece no sentido de oferecer suporte técnico no desenvolvimento de novas rotinas de acesso aos dados, ou ainda assessorar o usuário no aspecto da tecnologia envolvida no armazenamento dos dados.

8.4 Restrições de Segurança e Integridade

O controle do acesso dos usuários aos dados, através da verificação e determinação de permissões é uma das atividades fundamentais no papel a ser desempenhado pelo administrador do banco de dados. A garantia da integridade dos dados através da satisfação das propriedades ACID, é de fundamental importância para a segurança e confiança dos dados mantidos pelo SGBD. Alguns desenvolvedores preferem manter as restrições de integridade através de rotinas desenvolvidas por suas aplicações, no entanto o SGBD dispõe de ferramentas específicas e otimizadas para garantir esta integridade dos dados no nível do servidor de dados, o que sem dúvida alguma otimiza o processo de consulta aos dados.

8.5 Manter a disponibilidade dos dados

A disponibilidade dos dados é fundamental para o negócio de qualquer empresa que utilize um servidor de banco de dados, desta forma, esta é uma das atividades fundamentais na rotina de um administrador de banco de dados. A implementação de rotinas de backup's e seu conseqüente teste e a implementação de rotinas e mecanismos para a recuperação de falhas nos bancos de dados são tarefas essenciais na administração de um banco de dados.

8.6 Monitoração e otimização do desempenho.

Após a definição e "montagem" do banco de dados é necessário que se monitore o desempenho do sistema, a utilização do banco de dados irá alterar o preenchimento de algumas estruturas de dados como os índices, desta forma, a reorganização dos índices é uma das tarefas de otimização que podem ser incluídas na rotina de administração de uma banco de dados. A monitoração é fundamental para que tarefas de otimização de desempenho tenham local na rotina do administrador do banco de dados.

Referências Bibliográficas

DATE, J. C., **Introdução a Sistemas de Bancos de Dados, tradução da 7ª. edição americana,** Rio de Janeiro. Campus, 2000.

MOLINA, H. G.; ULLMAN, J. D.; WIDOM, J. **Implementação de Sistemas de Bancos de Dados**. Rio de Janeiro, Campus, 2001.

OZU, M. Tamer; VALDURIEZ, Patrick. **Principles of distributed databases systems.** New Jersey: Ed. Prentice-Hall, 1999.

PITOURA, E.; SAMARAS, G. **Data management for Mobile Computing,** Kluwer Academic Publishers, 1998.

SILBERSCHATZ, Abraham., KORTH, Henry F., SUDARSHAN S., **Sistema de Banco de Dados 3ª. edição ,** São Paulo. MAKRON Books Ltda, 1999.