Aula 02

Arquitetura Cliente-Servidor: Há um hospedeiro sempre em funcionamento, denominado servidor, que atende a requisições de muitos outros hospedeiros, denominados clientes.
Restrições (Constraints): Objetivo - Manter a consistência e Integridade do Banco de Dados

Aula 03

Ameaças aos Bancos de Dados:

- **Perda de Integridade:** a integridade é perdida se mudanças não autorizadas forem feitas nos dados.
- **Perda de Disponibilidade:** se um usuário ou um programa perde o acesso dos dados.
- **Perda de Confidencialidade:** se pessoas não autorizadas tem acesso aos dados, ou algum dado seja exposto sem autorização.

Segurança x Integridade:

- **Segurança:** refere-se a segurança contra acessos maldosos.
- **Integridade:** refere-se ao ato de evitar a perda acidental de consistência.

A fim de proteger o banco de dados, medidas de segurança precisam ser tomadas em diversos níveis:

- **Nível Físico:** o local onde o sistema de computador está localizado deve ser protegido contra assaltos ou intrusos. **Bom fornecimento de Energia, bom acondicionamento, equipe de monitoramento, backup.**
- **Nível Humano:** os usuários devem ser cautelosamente autorizados.
- **Nível de Sistema Operacional:** a fraqueza a nível de Sistema Operacional pode servir como meio de acesso não-autorizado ao banco de dados. Deve-se melhorar a segurança a nível de software.
- **Nível de Sistema de Banco de Dados:** alguns usuários possuem acesso a uma porção limitada do Banco de Dados

Criação de Usuários no PostgreSQL:

CREATE USER ‘flavio’ WITH PASSWORD ‘senha’;

O usuário criado não possui acesso a nada no banco, apenas pode se conectar. Para ter acesso, precisa-se ganhar privilégios. Para conceder privilégio a um usuário, utiliza-se o comando GRANT.

GRANT ALL ON DATABASE northwind TO flavio;
(Possui todos privilégios de Database: Create, Connect, Temporary)
GRANT ALL ON SCHEMA northwind TO flavio;
(Possui todos privilégios de Schema)
GRANT ALL ON TABLE northwind.customers TO flavio;
(Possui todos os privilégios na tabela Customers)
GRANT SELECT(id,name), UPDATE(id, name) ON nomes TO flavio;
(Possui acesso a Select e Update as colunas id,name na tabela nomes)

O comando WITH GRANT OPTION permite que o usuário criado possa conceder privilégios iguais ou menores, a outros usuários.

GRANT ALL ON TABLE northwind.customers TO flavio WITH GRANT OPTION;

O comando REVOKE permite aos administradores remover usuários e privilégios dos usuários.

REVOKE ALL ON northwind.customers FROM flavio;

REVOKE ALL ON DATABASE northwind FROM flavio;

Roles:

Para evitar que haja complicações na concessão e remoção de privilégios, é recomendado criar um grupo de usuários, e atribuir os privilégios a eles, ao invés de usuários separados. Esse grupo de usuários é chamado de Roles.

CREATE ROLE alunos;
CREATE USER 'flavio' WITH PASSWORD 'senha' IN ROLE alunos;
GRANT ALL ON SCHEMA northwind TO alunos;
GRANT SELECT ON northwind.orders TO alunos;

View x Grant:

O Grant garante um corte vertical na tabela, permitindo ao usuário manipular, no mínimo, uma coluna de uma tabela. Para permitir que o usuário tenha acesso ao um número específico de registros, é preciso combinar a View com o Grant.

- Cria-se uma View, e conceda acesso a usuário APENAS a essa View.

Exemplo de View:

CREATE VIEW departamento AS
(SELECT id, name
FROM departamento
WHERE codDept = 100);

CREATE MATERIALIZED VIEW mat_departamento AS
(SELECT id, name
FROM departamento
WHERE codDept = 100);

REFRESH MATERIALIZED VIEW mat_departamento;

DROP VIEW departamento;
DROP MATERIALIZED VIEW mat_departamento;

Aula 04

Modelo Fisico: a organização de registros em um arquivo, pode ser organizada de 3 maneiras:

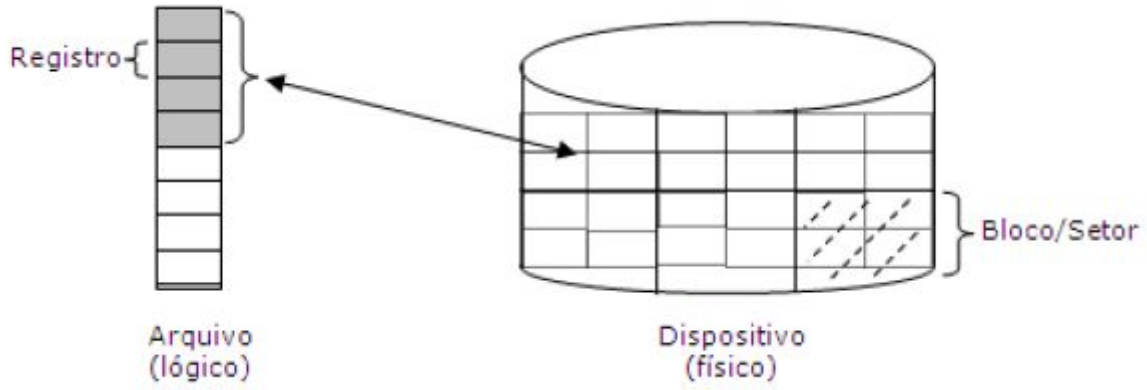
- **Organização de arquivos em Heap:**
  - Heap é sinônimo de aleatório, desordenado;
  - Um registro pode ser colocado em qualquer lugar no arquivo onde exista espaço para acomodá-lo;
  - Não existe ordenação dos registros dentro do bloco;
  - Normalmente existe um único arquivo para cada relação.
  - Existe apenas índice secundário, mesmo para a chave primária
- **Organização Sequencial de Arquivos:**
  - Os registros são armazenados em ordem sequencial, de acordo com o valor da "chave de busca" de cada registro;
  - Em geral, a chave de busca é a chave primária da relação.

- Para permitir uma recuperação rápida de registros na ordem da chave de busca, os registros são encadeados por ponteiros.
- O ponteiro em cada registro aponta para o próximo registro na ordem da chave de busca.
- Vantagem: Acesso sequencial de um conjunto de dados é o mais rápido. Útil quando é preciso processar quase todos os elementos.
- Desvantagens: Inclusão e Exclusão é extremamente custosa.
- Existe índice primário (esparso) para a chave primária da tabela
- Existe índice secundário (denso) para os demais campos

- **Organização de arquivos em Hashing:**
  - Uma função de Hash é calculada sobre algum atributo de cada registro;
  - O resultado de uma função de hashing especifica em que bloco do arquivo o registro deve ser colocado.
  - Existe apenas índice secundário, mesmo para a chave primária

Registro Lógico: corresponde às instâncias do mundo real, cujos atributos são representados pelos campos. Estes registros é que são manipulados pelo programa.
Registro Físico: corresponde à organização física onde as informações são armazenadas.

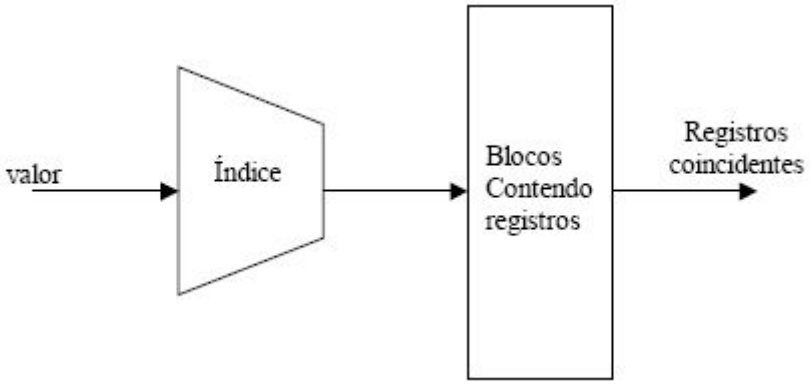
Organizar um arquivo significa relacionar(mapear) a estrutura lógica e a estrutura física de um arquivo.



A organização de um arquivo tem como objetivo **agilizar o processo armazenamento e recuperação de dados.**

Cada tabela é armazenada como um array de páginas (blocos) de tamanho fixo (normalmente 8kb). Na tabela, todas as páginas são logicamente equivalentes (mesma estrutura).
A estrutura utilizada para armazenar uma tabela é um **heap file**.

Index: Um índice permite localizar um registro sem ter que examinar mais que uma pequena fração dos registros possíveis. O(s) campo(s) cujos valores o índice se baseia formam a chave de pesquisa. Índices são, portanto, estruturas de dados auxiliares cujo único propósito é tornar mais rápido o acesso a registros baseados em certos campos, chamados campos de indexação.



Tipos Básicos de Índices:

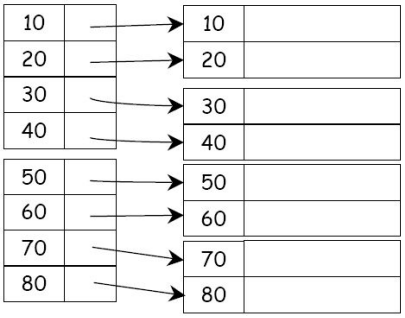
- Ordenados: Baseiam-se na ordenação dos valores.

- Hash: Baseiam-se na distribuição uniforme dos valores determinados por uma função (hash)

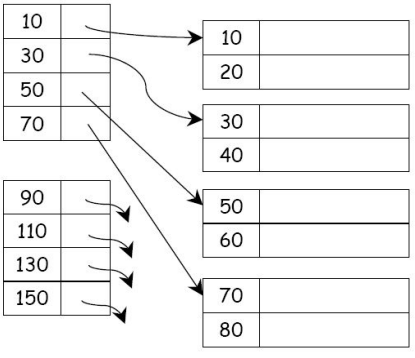
Classificação de Índices Ordenados:

- Quantidade de Entradas:

- Denso: Há uma entrada no índice para cada valor de chave que ocorre em um registro de dados.



- Esparso: Há um entrada no índice apenas para alguns valores de chave (um para cada bloco). Para localizar um registro com chave K, procura-se a entrada E do índice com o maior valor de chave menor ou igual a K e pesquisa-se o arquivo a partir do registro apontado por E.



- Um índice esparso utiliza menos espaço para armazenamento ao custo de um tempo um pouco maior para encontrar um registro dada sua chave.

- No índice esparso, inserções e remoções são menos custosas quando comparadas ao índice denso.

- Organização do Arquivo:

- Primário: No arquivo sequencial, um campo chave é usado para ordenar fisicamente os registro de arquivos em disco. Se a chave de ordenação for um campo exclusivo (sem duplicatas), então o índice é chamado de Índice Primário. A chave de busca de um índice primário é usualmente a chave primária.

- Vantagens: Economia de acesso aos blocos, busca binária.
- Desvantagem: Inclusão e Remoção são custosas.
- Índices Primários em geral são esparsos!

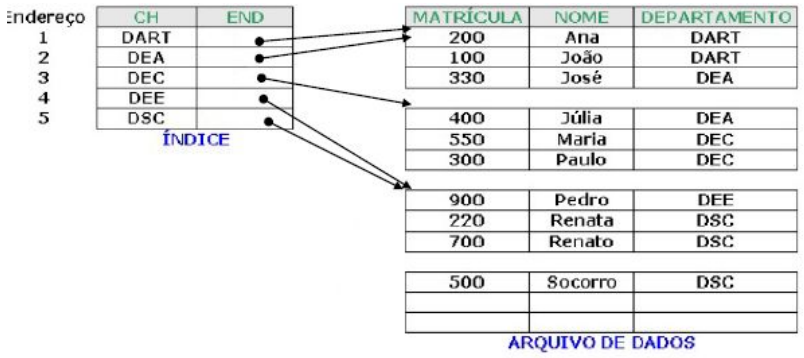
- Clustering: Se a chave de ordenação for um campo não chave (com duplicatas), então o índice é chamado de Índice Clustering. Existe uma entrada no índice clustering para cada valor distinto do campo clustering.

- O índice contém o valor e um ponteiro para o primeiro bloco no arquivo de dados que possua um registro com aquele valor para seu campo clustering.

- No Índice de Cluster os registros de um arquivo são fisicamente ordenados por um campo de ordenação que permite duplicatas.

- Índices de clustering são esparsos!

- O número de entradas é igual ao número de valores distintos do atributo de clustering.



Um arquivo pode possuir, no máximo, um índice primário ou um índice de clustering, porém não ambos. Porque um arquivo pode possuir, no máximo, um campo de ordenação física.

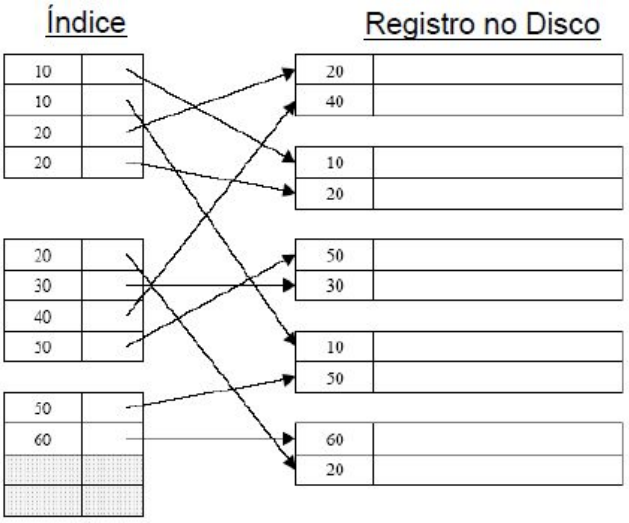
- Secundário: Frequentemente desejamos ter vários índices em uma relação, a fim de facilitar consultas variadas. Um terceiro tipo de índice, chamado de índice secundário, pode ser especificado sobre qualquer campo não-ordenado de um arquivo. Um arquivo pode possuir diversos índices secundários, além de seu método de acesso principal.

- Um índice secundário é um índice cuja chave não é aquela da ordem sequencial do arquivo.

- Portanto, a diferença entre índice secundário e índice primário está no fato de que o índice secundário não determina a colocação de registros no arquivo de dados.

- Índices secundários são sempre densos!!!

- Um índice secundário é um índice denso, normalmente com duplicatas. O índice consiste em pares chave-ponteiro. Os pares no arquivo de índices são classificados pelo valor da chave.

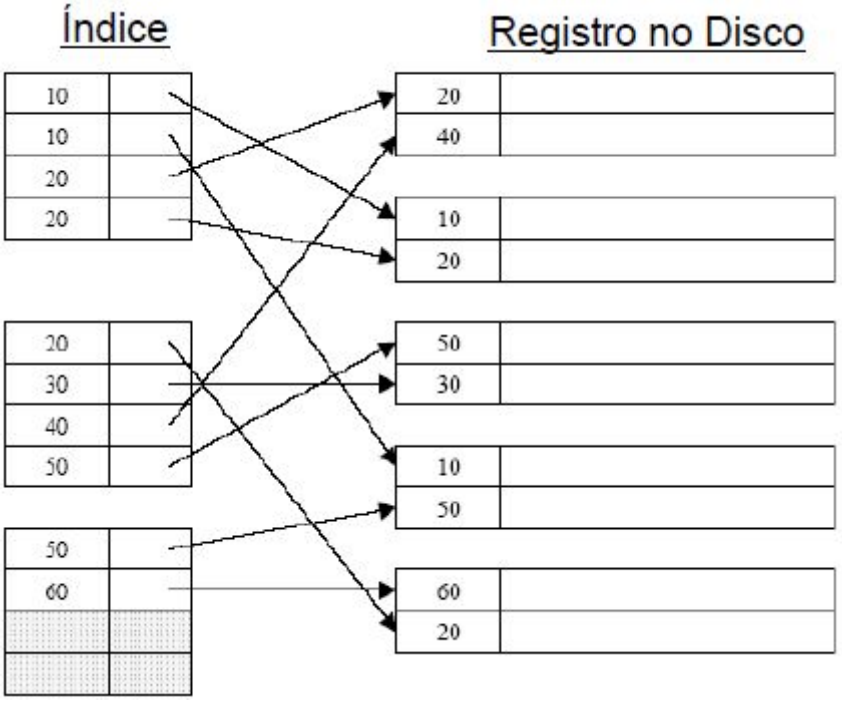


- Os dados não estão classificados pela chave de pesquisa.
- As chaves no arquivo de índices estão classificadas.

Aula 05

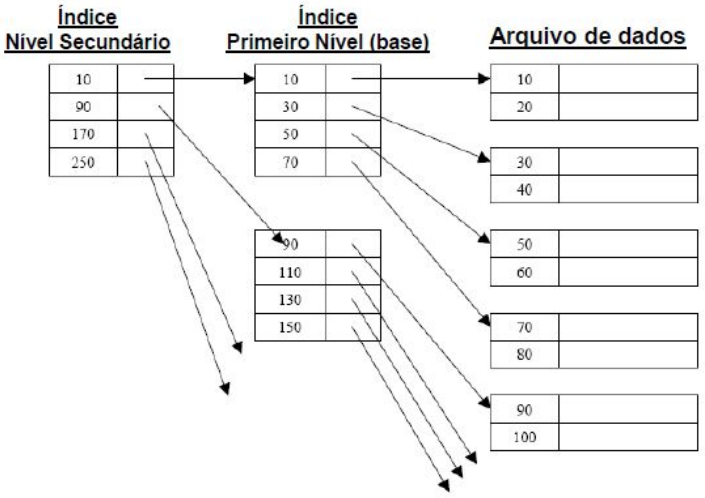
- Níveis de Indirecionamento:

- Mononível: Todos os índices visto até agora possuem apenas um nível de indireção.



Um índice pode cobrir sozinho muitos blocos (índices muito grandes). Se esses blocos não estiverem em algum lugar onde saibamos que é possível encontrá-los, por exemplo em cilindros designados de um disco, então talvez seja necessária outra estrutura de dados para localizá-los.

- Multinível: Inserindo um índice no índice, poderemos tornar o uso do primeiro nível de índices mais eficiente.



- Dá-se o nome de fator de bloco do índice (bfri) para o número de registros lógicos que cabem em um registro físico.
- O valor de bfri é chamado de fan out (fo) do índice multinível.

Embora um ou dois níveis de índices sejam com frequência muito úteis para acelerar consultas, há uma estrutura mais geral usada comumente em SGBDs comerciais por serem mais flexíveis e escaláveis.
A família geral de estruturas de dados é chamada árvore B, e a variante utilizada com mais frequência é conhecida como árvore B+.
Basicamente, a Árvore B:

- Automaticamente mantém os níveis balanceados para a quantidade de dados que está sendo indexada;
- Gerencia o espaço usado por seus blocos para que eles sempre estejam ocupados com pelo menos a metade de sua capacidade.

Árvore B:

- As árvores B são árvores balanceadas que visam otimizar as operações de entrada e saída nos dispositivos.
- Em uma aplicação comum de uma árvore B, a quantidade de dados é tão grande que provavelmente não caberia na memória principal.
- A árvore B copia blocos específicos para a memória principal quando necessário e os grava no disco se os blocos tiverem sido alterados.
- Na árvore B, uma chave somente é entrada uma vez em algum nível da árvore.
- Vantagens: Ausência de armazenamento redundante de chaves de busca;
- Possibilidade de encontrar uma chave sem chegar até um nó folha (busca mais rápida).

Árvore B+:

- Já na árvore B+, todos os dados só são armazenados nas folhas.
- Desta maneira, a estrutura conceitual das folhas difere da estrutura dos nós internos.
- As folhas da árvore B+ estão ligadas em sequência, tornando possível o acesso ordenado a seus campos.
- Vantagens: Embora a inserção e remoção em árvore B+ sejam complicadas, elas requerem relativamente poucas operações.

- É a velocidade de operações em árvores B+ que as torna uma estrutura de índice usada frequentemente em implementações de bancos de dados.
- Mecanismo para percorrer sequencialmente o arquivo de registros de dados sem que seja necessário visitar toda a árvore e ordenar o arquivo de registro de dados.
- Nó folha e não-folha são do mesmo tamanho. Facilita o gerenciamento do armazenamento para o índice;
- A remoção é mais simples, pois a entrada a ser removida sempre estará numa folha.

As vantagens da árvore B acabam quando os índices são muito grandes.
Assim, a simplicidade estrutural de uma árvore B+ é preferida por muitos implementadores de sistemas de banco de dados.

Aula 06

Analyze: Fornece uma análise de custo de tempo que uma consulta irá necessitar para ser realizada.
Explain Analyze: Oferece informações mais acuradas, pois realmente executa os comandos fornecidos.

Tipos de Índice no PostgreSQL:

Índice sobre a Chave Primária: criado automaticamente sempre que uma chave primária é definida.
Índice Secundário: qualquer índice criado no PostgreSQL
- Chave Unique cria índice secundário de forma automática.

Índice Composto: um índice composto por mais de um atributo (em geral 2, no máximo 3)
- O índice pode ser utilizado na consulta com os 2 atributos (and) ou na consulta do primeiro atributo.

Índices Parciais: construído sobre uma parte da tabela, definido por uma expressão condicional.

Índices Diretos:

Hash Estático: Uma função Hash ideal distribui as chaves armazenadas uniformemente por todos os buckets, de forma que todos os buckets tenham o mesmo número de registros.
- Pior função: mapeia todos os valores de chave de busca para o mesmo bucket.
- Função ideal: distribui as chaves armazenadas uniformemente por todos os buckets, distribuição aleatória.

Hash Dinâmico:
- A necessidade de fixar o conjunto B de endereços de buckets é um problema sério com a técnica de hashing estático. A maioria dos bancos crescem muito com o tempo.
- Divide e une os buckets enquanto o banco de dados cresce e encurta
- Eficiência do espaço é mantida
- Função gera valores por intervalos relativamente grandes
- Buckets criados por demanda

Seq Scan: toda vez que tem Select * sem filtro, será set scan, mais barato para recuperar do disco sem passar por uma estrutura de índice. Vai direto no arquivo da tabela e recupera para a memória principal.

Index Scan: supondo 5 registros de employeeid = 15, para cada vez que o 15 aparece na árvore, ele vai acessar o índice e para cada entrada que ele encontra, ele vai e acessa o disco para recuperar o dado. Para cada entrada que tiver no índice, ele acessa o disco quantas vezes forem necessárias.

Index Only Scan: significa que ele não vai na tabela, ele responde a consulta direto do índice. Ele procura o registro no índice, como não encontra, retorna dizendo que não encontrou o valor e não procurar o dado no disco. Se ele consegue responder a consulta direto pelo índice, ele não vai no arquivo, vai responder direto do índice.

Bitmap: ele não vai no registro, ele retorna o endereço do bloco onde os registros estão, e ordena os endereços dos blocos. Quando possui muitos registros retornando, fica mais barato retornar o bloco do que acessar cada índice.

Bitmap Index Scan: retorna os blocos

Bitmap Heap Scan: olhou no bloco, a condição é verdadeira, retorna o bloco.

Bitmap And: Bitmap Index Scan retorna os blocos que a condição é verdadeira, Bitmap And aplica recheck sobre os bitmaps para retornar onde os blocos são 1&1.

Bitmap Or:

Hash Agregate: tabela temporária que cria para fazer computação de agregação, joga todo mundo lá e depois conta.

Nested Loop: operação de junção, join, usa a informação que vem do índice para juntar as tabelas e mostrar o resultado.

Aula 07

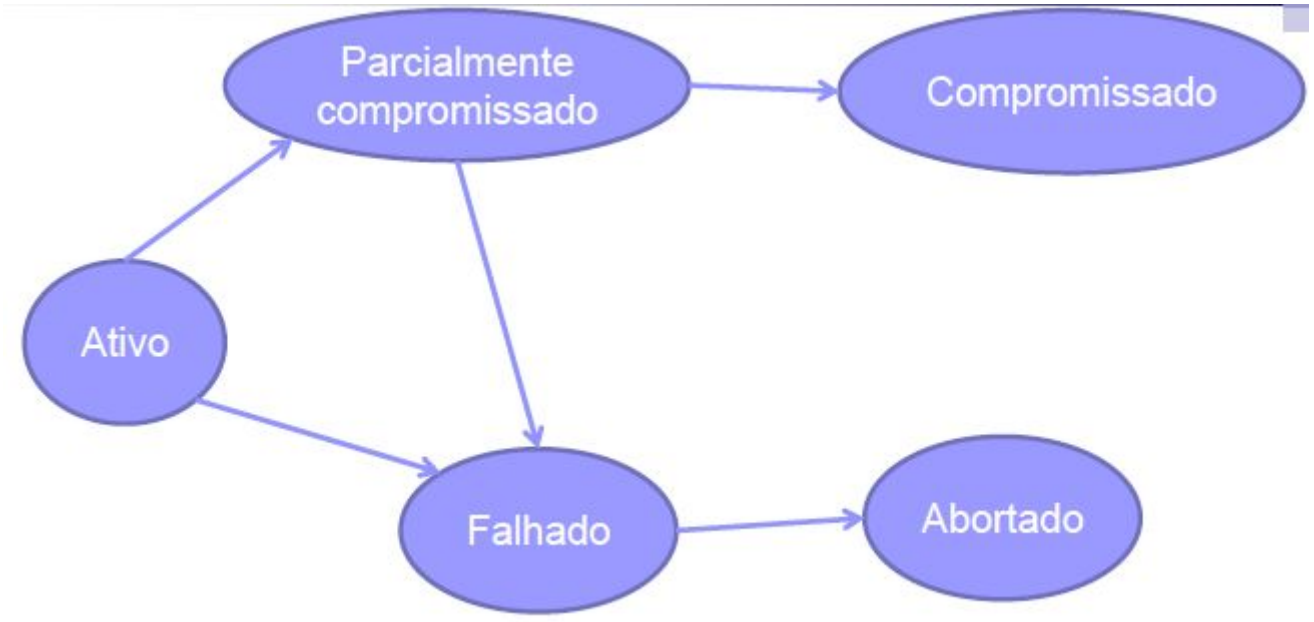
Transação é uma unidade lógica de trabalho, envolvendo diversas operações de bancos dados.
Está envolvida com os comandos Update, Delete e Insert. Select não está envolvido, apenas comandos de atualizações.

A integridade de uma transação depende de 4 propriedades, conhecidas como ACID:

- Atomicidade: ou tudo é feito, ou nada é feito. Ou todas transações concluem, ou voltam ao estado anterior.
- Consistência: é a validação de que o dado está íntegro e condiz com a realidade.
- Isolamento: conjunto de técnicas que tentam evitar que transações paralelas interfiram nas outras.
- Disponibilidade: garantir que o dado está disponível em casos de queda de energia, travamento.

Estados de uma Transação:

Estado Falhado: Se uma transação entrar em um estado de falha, ela tem duas opções: ou reinicia a transação, ou aborta a transação.
Estado Parcialmente Compromissado: Uma transação atinge esse estado, quando ela completou sua última instrução, mas ainda é possível abortá-la.
Estado Compromissado: também conhecido como “comitted”. Uma transação será compromissada se já estiver Parcialmente Compromissada e garantir que nunca será abortada, garantindo sua atomicidade e sua durabilidade.



Commit: operação de confirmação de que correu tudo bem e que todos os comandos que fazem parte da transação foram executados com sucesso e o banco de dados encontra-se em um estado consistente.

- Salva as alterações da transação no Disco, ao invés de deixá-las na memória principal.

Transações Implícitas: são aquelas que já tem um commit interno e todas alterações são salvas diretamente no disco.
Transações Explícitas: são aquelas que tem um ínicio e fim explicitado pelo desenvolvedor.

Por padrão, os SGBD's já executam um AUTOCOMMIT. Isso significa que se você executa uma instrução que modifica o banco, ela é alterada diretamente no disco.

Todas as alterações nas transações são temporárias.
Modificações são persistidas apenas após o **commit**.
A qualquer momento antes do commit as modificações podem ser canceladas através de um **rollback**.
O **rollback** não desfaz comandos da DDL, como CREATE TABLE, ALTER TABLE, etc.

START TRANSACTION
INSERT INTO departamento VALUES (3, "TI", 6, '2008-01-01');
UPDATE empregado SET salario = (salario*1.3);
ROLLBACK (todas alterações feitas após a transaction são restauradas)

START TRANSACTION
INSERT INTO departamento VALUES (3, "TI", 6, '2008-01-01');
UPDATE empregado SET salario = (salario*1.3);
COMMIT (todas as alterações feitas após o transaction foram salvas no disco)

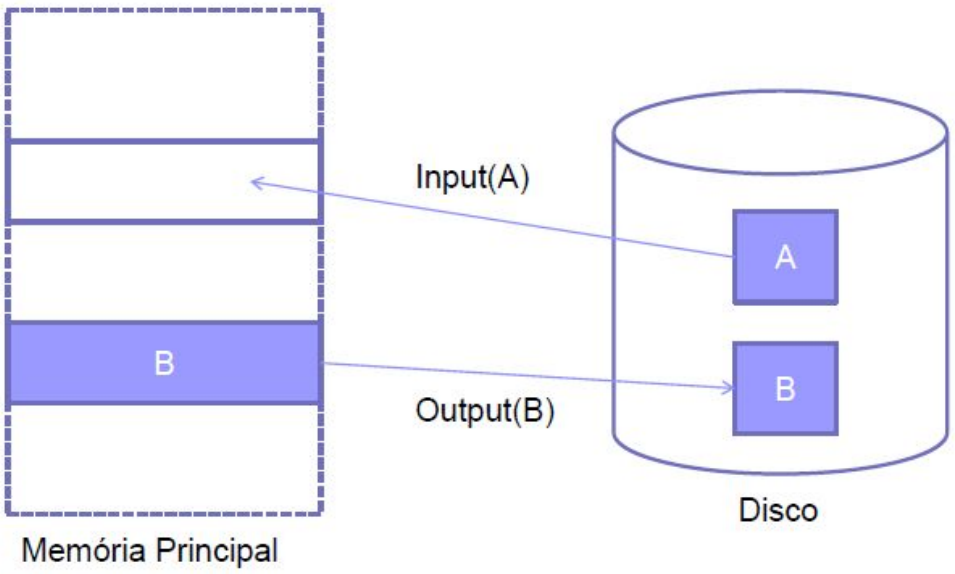
Savepoint: permite criar pontos de salvamento em meio a uma transação, permitindo que seja recuperada até determinado ponto.

As transações transferem blocos de informação do disco para a memória principal, depois os devolvem para o disco.

Os blocos residentes no disco são chamados blocos físicos, e os residentes da memória, de blocos de buffer.

Os movimentos de bloco entre o disco e a memória são feitos por duas operações:

Input(x): transfere do disco para a memória o bloco físico X.
Output(x): transfere da memória para o disco o bloco de buffer X e substitui o bloco físico apropriado.

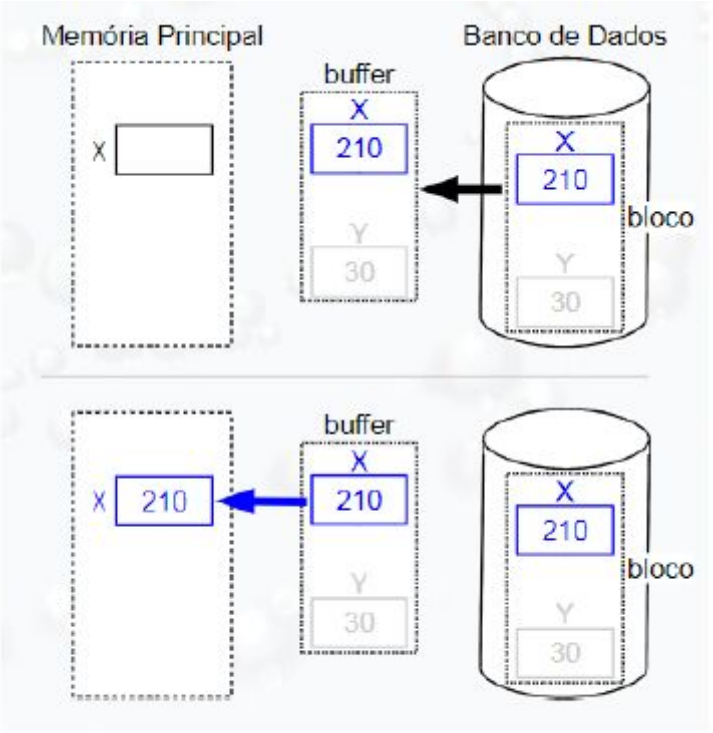


As transações interagem com o sistema de banco de dados transferindo dados de variáveis de programa para o banco de dados (arquivo!) e, do banco de dados para as variáveis de programa.

Essa transferência de dados é realizada por outras duas operações:

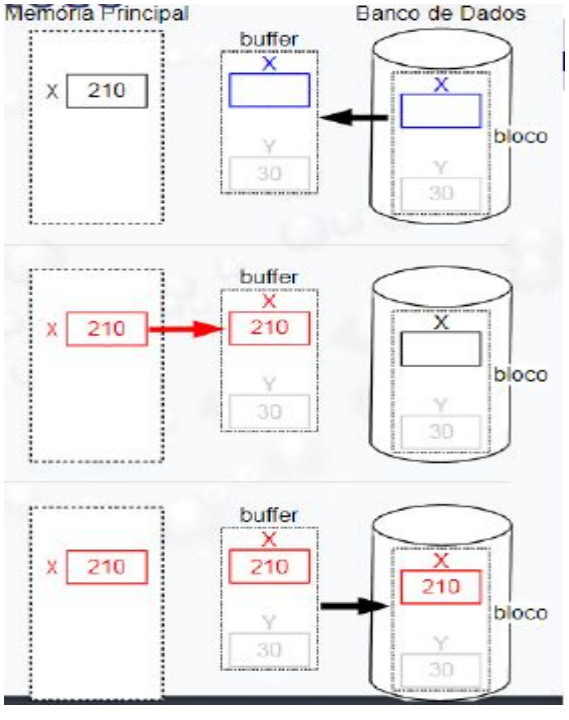
Read(X, xi): atribui o valor de X para a variável local xi.

- Se o bloco no qual X reside não estiver na memória, então emite input(X).
- Atribui o valor do bloco de buffer X para xi



Write(X,xi): atribui o valor da variável local xi para o bloco de buffer X.

- Se o bloco no qual X reside não estiver na memória, então emite input(X)
- Atribui xi para o valor de X do bloco de buffer



Exemplo:
UPDATE funcionarios SET salario = (salario * 1.3) WHERE matricula = 569871;

read(x
x := x + N
write(x)

Banco de Dados

Considere 1000 registros que cabem de 10 em 10 em um bloco de 4096 bytes

a) Qual o espaço necessário para os dados?

Bloco
10 registros
4096 bytes

1 bloco — 10 registros
x — 1.000.000 registros

x = 100.000 blocos

b) Se o campo da chave tem 32 bytes e um ponteiro tem 8 bytes:

- a) Quantos pares chave-ponteiro cabem em um bloco?

1 bloco = 4096 bytes
32 + 8 = 40 bytes: 1 par

$\frac{4096}{40} = 102,4 =$ 102 entradas de índice

b) Qual o espaço ocupado por um índice denso para esta relação?

1 bloco — 102
x — 1.000.000

x = 9803,92 blocos
x = 9803 blocos

c) E se um índice esparso for utilizado, qual o espaço gasto pelo índice?

1 bloco — 102
x — 100000

x = 980,39 blocos
x = 981 blocos

2- Considere um arquivo com 1000000 de registros de 167 bytes, armazenados num disco de tamanho de bloco igual a 8Kb:

a) Quantos blocos de disco são necessários para armazenar esse arquivo de dados?

1 registro = 167 bytes = 1 - 167b | 1 bloco - 49 reg
disco = 8 KB | x - 8192b | x/ - 1000000
x = 49 | x = 20408,16
x = 20409 blocos

b) Se para um dado atributo da tabela (índice secundário) ocupa 15 bytes, quantos blocos de disco são necessários para armazenar esse índice?

entrada de índice = 15 bytes | 1 - 15 | 1 b - 546
índice secundário = denso | x - 8192 | x - 1000000
x = 546 | x = 1831,5 blocos
x = 1832 blocos

c) Considerando que esse arquivo de dados seja um arquivo sequencial e que cada entrada de índice para a chave primária ocupa 12 bytes, quantos blocos de disco são necessários para armazenar o índice primário?

1 - 12 bytes | 1 bloco - 682 reg
x - 8192 | x - 20409 reg
x = 682 reg | x - 29,92
x = 30 blocos p/ índice primário