UNIVERSIDADE FEDERAL DO CEARÁ

Sistemas Gerenciadores de Banco de Dados

Angelo Brayner
Universidade Federal do Ceará
Departamento de Computação
brayner@dc.ufc.br

1. Database System
- Definition -

* Main Components
  * Database
    * Set of related data
  * Database Management System
    * Storage
    * Access
    * Concurrency control
    * Recovery (from failures)

1. Sistema de Banco de Dados

- Arquitetura de Três Camadas -

1. Database System
- Architecture -

1. Sistema de Banco de Dados

- Componentes da Arquitetura -

* SGBD
  * Compilador DML
    * Analisa sintaticamente e semanticamente comandos DML expressos em uma linguagem de consulta (ex. SQL)
    * Traduz estes comandos para uma das formas de representação interna de consultas (ex. álgebra relacional)
  * Interpretador DDL
    * Interpreta comandos DDL e os armazena no catálogo
      * Tabelas contendo meta-dados
      * Descrição do banco de dados
        * Esquema

1. Sistema de Banco de Dados

- Componentes da Arquitetura -

* SGBD
  * Mecanismo de Consultas
    * Responsável pela otimização e geração de planos de execução de consultas
  * Componente para SQL embutido
    * Pré-Compilador DML
      * Traduz comandos DML em chamadas a procedimentos (rotinas) na linguagem hospedeira
      * Até SQL Server 2000
        * nsqlprep precompiler

1. Sistema de Banco de Dados

- Componentes da Arquitetura -

* SGBD
  * Gerenciador de Transações
    * Controle de concorrência
    * Recuperação do banco de dados após falhas
  * Gerenciador de Buffer
    * Responsável por manter na área de buffer (memória principal) páginas mais acessadas
    * Política de alocação de páginas em buffer
      * LRU, MRU, FIFO, ARC
  * Gerenciador de Armazenamento
    * Responsável pelo armazenamento físico em memória secundária

1. Sistema de Banco de Dados

- Componentes da Arquitetura -

* BD
  * Arquivos de dados
    * Armazena os dados
  * Índices
    * Estruturas de índices para os arquivos de dados
      * Índices ordenados
        * Árvores B^+, Arquivos grade, Árvores kd, Árvores quadrantes
      * Índices não ordenados
        * Indice hash

1. Sistema de Banco de Dados

- Componentes da Arquitetura -

* BD
  * Catálogo
    * Armazena esquema do banco de dados (meta-dados)
      * Nomes das tabelas, Atributos de cada tabela, Definição de índice para uma tabela, etc...
    * Armazena informações estatísticas
      * Cardinalidade de uma tabela, fator de seletividade de colunas, etc
      * Utilizadas na otimização de consultas
  * Fragmentos de código
    * Stored procedures, Triggers, User defined functions

1. Database System
- Logical DB Structure -

* Oracle

1. Database System
- Logical DB Structure -

* SQL Server

1. Database System
- Logical DB Structure -

* Database management system accesses pages from the database
  * Page is a set of tuples
  * Access unit for DBMS
* A table may be stored in several pages

1. Sistema de Banco de Dados

- Classificação -

* Modelo de Dados
  * Sistema de Banco de Dados Relacional
    * Modelo Relacional
    * Tipo primitivo de dados
      * Relação (tabela)
        * Conjunto de tuplas (linhas)
    * Conjunto de Operadores
      * Álgebra e Cálculo Relacional
    * Restrições de integridade
      * Integridade de chave primária, integridade referencial, restrição de domínio

1. Sistema de Banco de Dados

- Classificação -

* Modelo de Dados
  * Sistema de Banco de Dados Orientado a Objeto
    * Modelo orientado a objeto
      * Objeto, herança, composição, métodos
    * Tipos primitivos
      * Objeto, conjunto, lista, string, integer, real
  * Sistema de Banco de Dados Objeto-Relacional

1. Sistema de Banco de Dados

- Classificação -

* Arquitetura
  * Centralizada
    * Sistema de Banco de Dados Centralizados
      * Os componentes do SBD residem no mesmo host
  * Distribuída
    * Sistema de Banco de Dados Cliente-Servidor
      * Distribuição de funções do SGBD entre clientes e servidor
    * Sistema de Banco de Dados Paralelos
      * Distribuição do controle de funções do DBMS entre diversos sistemas computacionais
    * Sistema de Banco de Dados Distribuídos
      * Distribuição de dados através de SBDs homogêneos
        * Fragmentação
        * Replicação

1. Sistema de Banco de Dados

- Classificação -

* Arquitetura
  * Distribuída
    * Sistema de Banco de Dados Heterogêneos
      * Distribuição de dados através de SBDs heterogêneos
        * Sistema de banco de dados múltiplos (MDBS),
          Sistema de banco de dados federados (FDBS),
          Arquitetura de Mediadores
    * Sistema de Banco de Dados Móvel
      * Distribuição de funções e dados do SGBD entre clientes e servidor em ambiente de computação móvel
    * Sistema de Fluxo de Dados
      * Distribuição de funções e dados entre componentes que geram e acessam fluxos de dados
      * Redes de Sensores Sem Fio

1. Sistema de Banco de Dados

- Classificação -

* Arquitetura
  * Distribuída
    * Sistema de Banco de Dados Multi-plataforma
    * Replicação de dados sob demanda em múltiplos dispositivos

2. Memory System

* Computer system
  * Requires a memory system
    * To store data and instructions
* Components
  * Storage devices
    * Recording medium
      * Physical material on which data is stored in recoverable format
    * Magnetic disk, DRAM, magnetic tape, punch card
  * Set of algorithms
    * To access (w/r) the devices

2. Memory System

* Processors
  * May execute 128×10^9 instructions/second (Intel i7 2600k)
    * ~7.8×10^{-12}s to execute 1 instruction
* Access time - Main Memory
  * 7×10^{-9}s
* In order to process 2 instructions
  * A processor may wait ~6.9922×10^{-9} s
    * The second instruction should be fetched
* To reduce CPU wait time
  * Storage (Memory) Hierarchy

2. Memory System

* Storage Hierarchy

2. Memory System

* For each storage hierarchy level
  * Specific storage devices
    * Main memory
      * DRAM
        * Stores each bit of data in a capacitor within an integrated circuit
    * Secondary memory
      * Hard Disk, Solid State Memory
    * Tertiary memory
      * Optical disks
      * Magnetic tape

2. Memory System
- Memory Classification -

* Data persistence
  * Volatile
    * Requires power to persist data
    * Registers, Cache and Main memory
  * Non-volatile
    * Holds stored data regardless energy supply
    * Hard disks, magnetic tapes, optical disks

2. Memory System
- Memory Classification -

* Access method
  * Random - RAM
    * Any memory address is accessed in the same amount of time
    * Registers, Cache and Main memory
  * Sequential - SAM
    * Data stored in different memory addresses may have different access time
    * Hard disks

2. Memory System

- Memory Types [Registers] -

* Small amount of storage available as part of a processor (CPU) →
  * Size (word)
    * Multiple of 8 bits: [8, 64]
* Types
  * Memory Buffer Register (MBR)
    * Holds data being transferred from cache (L1) to processor and vice versa.

2. Memory System
- Memory Types [Registers] -

* Types (cont.)
  * Memory Address Register (MAR)
    * Stores memory address (L1) of data to be transferred to register or the address to which data will be moved in L1
  * Instruction Register (IR)
    * Holds the instruction currently being executed
* Access time
  * ~1ns (10^{-9}s)
* Physical storage medium
  * Metal Oxide Semiconductor (MOS)

2. Memory System

- Memory Types [Cache] -

* Memory between Registers and Main Memory
  * To reduce the processor wait time for accessing data/instruction from main memory
* Often, divided into two levels
  * L1
    * Located on the the same chip as the processor
    * 16k
  * L2
    * Located on a different chip
    * 2MB

2. Memory System
- Memory Types [Cache] -

* Physical storage medium
  * Static RAM (SRAM)
    * Semiconductor memory
    * Stores a bit
      * Bistable latching circuitry
      * Four transistors
    * Does not need to be periodically refreshed
  * Access time
    * ~10ns

2. Memory System

- Memory Types [Main Memory] -

* Word-addressable storage
  * Controlled by the CPU
* Access unit (r/w)
  * Word
    * 16, 32, 64 bits
* Physical Storage Medium
  * Dynamic RAM (DRAM)
    * Semiconductor memory
      * Composed of cells
        * Cells of 8 bits
      * Thus, in a memory of 64 bits
        * An address for a set of 8 cells
  * Access time
    * ~60ns

2. Memory System
- Memory Types [Main Memory] -

* Connected to CPU by 2 memory buses
  * Address bus
  * Data bus

2. Memory System

- Memory Types [Secondary Memory] -

* Non-volatile memory
  * Used to persist application data
* Orders of magnitude slower and more capacious than main memory
* Access time to get the contents of a given memory address depends on its location on the device
  * Accessing data stored on the first track of a magnetic disk is much more faster than accessing data on the last track

2. Memory System

- Memory Types [Secondary Memory] -

* Used Medias for secondary storage
  * Punched cards

Punch card machine: Storing data on punched cards

IBM 711 card reader on an IBM 704 computer at NASA (1957)

2. Memory System

- Memory Types [Secondary Memory] -

* Used Medias for secondary storage
  * Magnetic tape
    * Long strip of plastic film overlaid by a thin magnetizable coating

2. Memory System

- Memory Types [Secondary Memory] -

* Used Medias for secondary storage
  * Hard (magnetic) disks
    * IBM 305 RAMAC (1956)
      * First computer with 5MB HDD
        * HDD weighted over a ton
    * Samsung Spinpoint M9T
      * 2.5-Inch HDD
      * 2TB
      * 5400rpm

2. Memory System
- Anatomy of an HD Drive -

2. Memory System
- Anatomy of an HD Drive -

2. Memory System
- Anatomy of an HD Surface -

Surface (2 per platter)

- Covered with a thin layer of magnetic material
- Organized into tracks
  - Concentric circles

Track

- Circular path on disk surface
- Stores information (bits)
- Subdivided into Sectors

Sector

- Smallest physical storage unit on the disk
  - Read/Write unit
  - Indivisible unit as far as errors are concerned
    - Should a small region *R* of the magnetic coating be corrupted in some way
      - The entire sector containing *R* is corrupted as well
        - It can not store information anymore
- Each sector stores the same amount of data
  - 4096 Bytes

Gap

- non-magnetized area
- Separates sectors
- ~10% of the track

2. Memory System
- Anatomy of an HD Surface -

* Each disk sector is labelled using the factory track-positioning data
  * Sector identification data
    * Written to the area immediately before the contents of the sector
    * Identifies the starting address of the sector
* Block/Cluster
  * Logical unit
    * Used to transfer data between disk and main memory
  * Typically, consists of one or more sectors

2. Memory System
- Disk Controller -

* Processor
  * Selects the surface s from which to read/write
  * Controls the actuator to position the heads at a particular radius (track *t*)
  * Selects the sector from *t* of s, which is under the head
  * Transfers bits
    * Sector->Main memory (r)
    * Main memory->Sector (w)
  * Buffer
    * To flush burst of writes

2. Memory System
- Disk Latency -

* Time interval between the moment the command to access the disk is triggered and it is completed
* Seek time
  * Time to position the r/w head at the proper track
  * Speedup time
    * Time to accelerate the arm until it reaches a fixed maximum velocity or half of the distance to the proper track
      * Acceleration rate may reach 550g
  * Travel time (long seeks)
    * Time in which the arm moves at constant speed (max speed)

2. Memory System
- Disk Latency -

* Seek time (cont.)
  * Slowdown time
    * Time to bring the arm to rest close to the desired track
  * Settle time
    * Time to disk controller to adjust the head to position it at the correct track
  * In long seeks, travel time dominates seek time

2. Memory System
- Disk Latency -

* Rotational latency (RL)
  * Time for the disk to rotate in order to position the proper sector under the r/w head
  * On the average, time for half rotation
  * Let r be the amount of rotations per second
    * $RL = (2r)^{-1}$
* Transfer time (TT)
  * Time to transfer data
    * From the sector to the head (r)
    * From the head to the sector (w)

2. Memory System
- Disk Latency -

* Transfer time (cont.)
  * Let T be the track storage capacity and S the sector storage capacity in bytes
    * $TT = S(rT)^{-1}$
* Disk latency (DL) to access one sector
  * $DL = S_T + (2r)^{-1} + S(rT)^{-1}$

Exercise

* The X disk has the following characteristics:
  * There are 4 platters
  * Each surface has 10^5 tracks
  * Each track has 10^3 sectors of 512 bytes
  * 10% of each track represents the gaps
  * The disk rotates at 10^4 rpm
  * The time it takes the head to reach the track n is
    * $1.5+10^{-4}n$ ms
* Compute
  * The disk capacity
  * The maximum seek time

2. Memory System
- RAID -

* Redundância para aumentar a confiabilidade de HDs
  * Pode-se aumentar tempo médio de perda de dados através da redundância de dados
    * Armazenar dado redundante
      * Dado normalmente não necessário
      * Utilizado para reconstruir dado perdido após falha de disco
  * Exemplo
    * Espelhamento de discos
      * Caso um disco falhe
        * dados podem ser acessados no disco espelho
      * Perda total de dados
        * Ocorre se o disco espelho falhar antes que o primeiro seja recuperado

2. Memory System
- RAID -

* Distribuição de dados para aumentar desempenho de HDs
  * Acesso paralelo
  * Em um conjunto de N discos
    * Acesso paralelo a diversos discos através da distribuição paralela de dados nos múltiplos discos
      * Aumentar a taxa de transferência de dados
* Estratégias para distribuição de dados
  * Distribuição paralela de bits
    * Distribuir os bits de um byte em diferentes discos
      * Exemplo
        * Em um conjunto de oito discos, cada bit i é armazenado no disco i
        * Em um conjunto de quatro discos, como deverá ser realizada a distribuição de bits de um byte ?

Os bits i e 4+i de um byte são armazenados no disco i

2. Memory System
- RAID -

* Distribuição de dados para aumentar desempenho
  * Estratégias para distribuição de dados (cont.)
    * Distribuição paralela de blocos
      * Distribuir os blocos de um arquivo em diferentes discos
      * Política de alocação de blocos de um arquivo
        * Bloco i alocado no disco (i mod n)+1
        * n representa o número de discos disponíveis

2. Memory System
- RAID -

* Mecanismo de RAID
  * Combinar diferentes graus de distribuição de dados e de redundância
    * Níveis de RAID
  * Fatiamento (striping)
    * Concatenar múltiplas unidades de disco em uma única unidade de armazenamento lógica

2. Memory System
- RAID -

* RAID nível 0
  * Organização de discos com distribuição paralela
    * Baseada em blocos
      * Blocos de um arquivo são armazenados em diversos discos
  * Não apresenta redundância
  * Exemplo

2. Memory System
- RAID -

* RAID nível 1
  * Distribuição paralela
    * Baseada em blocos
  * Redundância implementada através de replicação
  * Equivale ao nível 10, existente no mercado
  * Exemplo
    * RAID 1 com quatro discos de dados e quatro discos para replicação

2. Memory System
- RAID -

* RAID nível 2
  * Organização de discos com distribuição paralela
    * Baseada em bytes
  * Redundância implementada através de bits de correção
    * Bits de correção podem ser implementados através de bit de paridade
      * Para cada byte armazenado existe um bit de paridade
        * 0
          * Se o número de bits com valor 1 é par (paridade 0)
        * 1
          * Se o número de bits com valor 1 é ímpar (paridade 1)

2. Memory System
- RAID -

* RAID nível 2
  * Estratégia do RAID nível 2 é
    * Implementar o código de correção de erros
      * Dois ou mais bits adicionais
    * Implementar distribuição paralela de bytes
      * Bits em diferentes discos
    * Organização de código de correção de erros

2. Memory System
- RAID -

* RAID nível 2
  * Exemplo

2. Memory System
- RAID -

* RAID nível 3
  * Semelhante ao nível 2
  * Controladoras armazenam bits de paridade por setor de disco
  * Bits de um mesmo byte num mesmo disco
    * Armazenados em setores diferentes
  * Menor número de discos para armazenar bits de correção

2. Memory System
- RAID -

* RAID nível 4
  * Distribuição por blocos
  * Redundância implementada através de blocos de paridade

2. Memory System
- RAID -

* RAID nível 4
  * Caso um disco falhe
    * Bloco de paridade mais os blocos dos discos em funcionamento são utilizados para recuperar os blocos do disco que falhou

2. Memory System
- RAID -

* RAID nível 5
  * Mecanismo de paridade distribuída de blocos entrelaçados
  * Distribuição
    * Blocos de dados e de paridade são distribuídos nos N+1 discos
    * Lembre-se que no RAID nível 4
      * bloco de paridade é armazenado em um disco separado
  * Redundância implementada através de blocos de paridade
    * Um bloco de paridade para cada bloco armazenado nos discos de dados

2. Memory System
- RAID -

* RAID nível 5
  * Para cada bloco i em um conjunto de n discos
    * Bloco de paridade de i é armazenado no disco (i mod n) +1
    * Os i-ésimos blocos dos outros quatro discos armazenam os dados do bloco I

2. Memory System
- RAID -

* RAID nível 5
  * Exemplo (5 discos)
    * Caso um disco falhe
      * Bloco de paridade mais os blocos dos discos em funcionamento são utilizados para recuperar os blocos do disco que falhou

2. Memory System
- RAID -

* RAID nível 6
  * Semelhante ao nível
    * Armazena informações adicionais para recuperar dados no caso de falha em mais de um disco
    * Garante maior confiabilidade que o nível 5

2. Memory System
- RAID -

* Escolha do nível de RAID
  * Quantidade de disco disponíveis
  * Tempo e complexidade da atividade de reconstrução de discos após falhas
    * Aplicações que requerem fornecimento contínuo de dados
      * RAID 1
        * Baixa complexidade e baixo tempo de reconstrução

2. Memory System
- RAID -

* Escolha do nível de RAID
  * Tipo de aplicações que acessam discos do mecanismo RAID
    * RAID nível 0
      * Aplicações de alto desempenho
      * Discos bastante confiáveis
    * RAID nível 1
      * Aplicações que requerem fornecimento contínuo de dados
    * RAID nível 5
      * Aplicações com fornecimento contínuo de dados e que armazenam grandes volumes de dados

2. Memory System
- Solid State Drive -

* Over the last 3 decades
  * The speed of processors has increased exponentially
  * The number of IOPS provided by HD has only increased marginally
  * There is a gap between HDD access time and the time processors can consume data
* A new class of storage devices
  * Solid-state Drive (SSD) or Storage Class Memory (SCM)
  * Data persistence (secondary memory)

2. Memory System
- Solid State Drive -

* Storage device made of silicon chips instead of spinning metal platters
  * No mechanical part

Fusion-io's ioMemory Module of 1.28TB

2. Memory System
- Types of Solid State Drive -

* Flash (NAND)
  * Computer chip which can be electrically reprogrammed and erased
  * Stores data in a floating-gate transistor (cell)
    * Array of cells
    * Data are represented by the voltage level in a cell
      * High voltage (>5v) - 0
      * Low voltage - 1
        * Default state
  * Block/page addressable
    * Read on a page
    * Write on a block

2. Memory System
- Types of Solid State Drive -

* Phase-change Memory (PCM)
  * Stores data in a chalcogenide glass (same material used in CD-RW and DVD-RW)
    * The heat produced by an electric current may induce two different states
      * crystalline (low resistance and cool state) - Bit 1
      * amorphous (high resistance and hot state) - Bit 0
    * Byte addressable
    * PCM has not yet reached a commercialization stage
      * IBM, Samsung, Toshiba, etc...

2. Memory System
- SSD Properties -

* Read/Write Asymmetry
  * Execution time

2. Memory System
- SSD Properties -

* Read/Write Asymmetry
  * Energy consumption
* Write endurance
  * Lifetime determined by the number of write operations
  * 2011
    * Flash
      * 10^5
    * PCM
      * 10^7

2. Memory System
- Anatomy of a NAND Flash -

2. Memory System
- Anatomy of a NAND Flash -

Package: (4, 6, 8, 16) Chips

Chip (Die): Planes (several)

Plane: Thousands of blocks (2048)

Block: several pages (64 2k-pages)

2. Memory System
- Anatomy of a NAND Flash -

2. Memory System
- NAND Flash Properties -

* Single Layer Cell (SLC)
  * One bit per cell
* Multi-Layer Cell
  * 2 bits per cell
    * 00 - High voltage
    * 01 - Medium high voltage
    * 10 - Medium low voltage
    * 11 - Low voltage

2. Memory System
- NAND Flash Properties -

* Single Layer Cell (SLC)
  * Faster
  * More reliable
* Multi-Layer Cell
  * Presents higher density

2. Memory System
- NAND Flash Properties -

* Operations
  * Read
    * Page addressable
  * Erase
    * Sets all bits to 1
    * Block addressable
  * Program (physical write)
    * Sets a bit to 0
    * It can only be executed on a "clean" block
      * A block with all bits set to 1
    * Page addressable

2. Memory System
- NAND Flash Properties -

* Once all blocks have at least a bit set to 0
  * A (logical) write operation has to be executed
    * Erase and program operations
      * An erase operation (block addressable)
        * To set all bits to 1
      * A program operation
        * Page addressable
* Erase and Program operations (logical write)
  * 2 orders of magnitude slower than a read
  * Logical writes should be avoided

2. Memory System
- NAND Flash Properties -

* Wear Leveling
  * Every time a block is written
    * Its lifetime is decreased
  * Wear leveling aims at minimizing this effect by swapping intensely-used blocks with rarely-used ones
  * This requires rewriting blocks, which is expensive

2. Memory System
- NAND Flash Properties -

* Garbage Collection
  * Responsible for "cleaning" blocks with stale pages

Erase
Sets all bits to 1

2. Memory System
- NAND Flash Properties -

* Write amplification
  * Both the garbage collection and the wear leveling cause extra writes on SSDs
  * The amount of actual physical writes on flash disks is thus much larger than the amount of logical writes required by applications

2. Memory System
- NAND Flash Properties -

* Write endurance
  * SSDs have lifetime determined by the number of erase/program cycles
    * Typically 100k cycles
* Low energy footprint
  * To perform read/write operations, only logical gates (circuitry) are involved;

2. Memory System
- NAND Flash Properties -

* Low random access time
  * Random access is at least a few orders of magnitude faster than HDDs
* High random IOPS rates
  * There is no mechanical seek or latency time to overcome
* Asymmetry of read and write execution time
  * A read operation is at least 2 orders of magnitude faster than a write operation

2. Memory System

- Memory Types [Tertiary Memory] -

* Images gathered by satellites
  * 1 image/2 min
  * Petabytes
* Big data storage
  * By 2020, global data storage will amount to 40 trillion GB
    * 5,200 GB per person
* There are databases much larger than what can be stored on disks
  * Not economically viable

2. Memory System

- Memory Types [Tertiary Memory] -

* Tertiary memory presents smaller cost per bit
  * Higher access time
  * Larger capacities
* Magnetic tapes
  * Sony tape
    * 185TB per cartridge
      * ≈ 3,700 Blu-ray discs
    * Dimensions
      * 102.0 × 105.4 × 21.5 mm

2. Memory System

- Memory Types [Tertiary Memory] -

* Oracle StorageTek SL8500 Modular Library System
  * http://www.oracle.com/us/products/servers-storage/storage/tape-storage/sl8500-modular-library-system/overview/index.html

2. Memory System
- Storage Manager -

* Gerenciador de Armazenamento (arquivo)
  * Componente do SGBD (SO) responsável pelo gerenciamento de arquivos
* Arquivos
  * Abstração que fornece uma maneira de agrupar dados logicamente relacionados
    * Torna transparente para usuários detalhes como:
      * Como e onde os dados estão armazenados
      * Detalhes físicos para acessar a mídia de armazenamento

2. Memory System
- Storage Manager -

* Arquivos
  * Não estruturado
    * Seqüência de bytes
    * Qualquer semântica deve ser definida a nível de programa
    * UNIX
  * Estrutura de registro
    * Seqüência de registros
    * Unidade para operações de leitura e escrita

2. Memory System
- Storage Manager -

* Armazenamento de Arquivos
  * Política de alocação de arquivos em disco
    * Identificar em quais blocos os dados de um arquivo estão armazenados
  * Diferentes políticas

2. Memory System
- Storage Manager -

* Armazenamento de Arquivos
  * Alocação Contínua
    * Armazenamento em blocos contíguos em disco
      * Considere um disco com blocos de 2K
        * Arquivo de 60K ocupa 30 blocos consecutivos
    * Vantagens
      * Facilidade em localizar todos os dados em disco
        * Basta saber o endereço do primeiro bloco que contem dados do arquivo
        * Acessar seqüencialmente esses blocos
      * Menor tempo de acesso aos dados em disco
    * Desvantagens
      * Reservar espaço para armazenar um arquivo
        * Arquivos crescem dinamicamente
        * Fragmentação

2. Memory System
- Storage Manager -

* Armazenamento de Arquivos (cont.)
  * Alocação com Lista Encadeada
    * Cada arquivo é alocado numa lista encadeada de blocos de disco
    * A primeira palavra de um bloco contem um ponteiro para o próximo bloco do mesmo arquivo

2. Memory System
- Storage Manager -

* Armazenamento de Arquivos (cont.)
  * Alocação com Lista Encadeada (cont.)
    * Vantagens
      * Facilidade em localizar todos os dados em disco
        * É suficiente guardar apenas o endereço do primeiro bloco
        * Acessar os outros blocos através da lista encadeada
      * Não ocorre fragmentação
    * Desvantagens
      * Acesso randômico
        * Lento
      * Identificação de blocos vazios (não utilizados)

2. Memory System
- Storage Manager -

* Armazenamento de Arquivos (cont.)
  * Alocação com Lista Encadeada Utilizando Índices
    * Alocação com lista encadeada
      * Ponteiros armazenados em tabela na memória principal
    * Exemplo
      * Empregados.mdf ocupa os blocos
        * 10, 7, 3, 11
      * Brayner.txt ocupa os blocos
        * 6, 2, 8

2. Memory System
- Storage Manager -

* Armazenamento de Arquivos (cont.)
  * Alocação com Lista Encadeada Utilizando Índices

Bloco atual	Próximo bloco
1	
2	8
3	11
4	
5	
6	2
7	3
8	Null
9	
10	7
11	Null

Brayner.txt → 6

Empregados.mdf → 10

2. Memory System
- Storage Manager -

* Armazenamento de Arquivos (cont.)
  * Alocação com Lista Encadeada Utilizando Índices
    * Vantagens
      * Facilidade em localizar todos os dados em disco
      * Não ocorre fragmentação
      * Todo o bloco é ocupado só com dados
      * Acesso randômico mais eficiente
    * Desvantagens
      * A tabela precisa ser carregada em memória
        * Disco de 1TB e blocos de 4.096 Bytes
          * Tabela com 244.140.625 entradas
          * Cada entrada ocupa em média 8 bytes
          * A tabela ocuparia ~1.9GB

2. Memory System
- Storage Manager -

* Armazenamento de Arquivos (cont.)
  * Alocação utilizando i-nodes
    * Cada arquivo está associado a uma tabela
      * Denominada de index-node (i-node)
    * Conteúdo de um i-node
      * Atributos do arquivo
      * Endereços dos blocos, onde estão armazenados os dados do arquivo

2. Memory System
- Storage Manager -

* Armazenamento de Arquivos (cont.)
  * Alocação utilizando i-nodes
    * Tabela i-node deve ser carregada na memória
      * Tabela de tamanho fixo
    * Solução para endereçar os blocos de disco ocupados por grandes arquivos no i-node
      * single indirect block
      * double indirect block
      * triple indirect block

2. Memory System
- Storage Manager -

* Armazenamento de Arquivos (cont.)
  * Alocação utilizando i-nodes

3. Buffer Management

3. Buffer Management

* Goal
  * To hold on main-memory (buffer pool) the most referenced pages
    * Reduces the amount of disk access
* Database Buffer Manager
  * Buffer replacement policy
  * Efficiency metric
    * Hit Ratio
      * Measures the probability of finding the desired page in the buffer pool

3. Buffer Management

* A taxa de acerto é determinada experimentalmente como descrito a seguir:
  * Um conjunto de programas (consultas) é executado
  * Os números de referências de endereços satisfeitas por M1 e M2 são registrados
    * Números denotados por N1 e N2
  * Taxa de acerto
    * H= N1 / (N1+N2)
  * A taxa de acerto ótima deve tender a 1
    * Porque?

3. Buffer Management

* Considere TA_1 e TA_2 o tempo de acesso de M_1 e M_2
* Tempo médio para o SGBD acessar um dado no sistema de memória é dado por:
  * $TA = H.TA_1 + (1-H).TA_2$ (1)
* Tempo de transferência da página (TB)
  * $TA_2 = TB + TA_1$ (2)
* Substituindo (2) em (1), temos:
  * $TA = TA_1 + (1-H).TB$

3. Buffer Management

* Buffer-page replacement policy
  * LRU (Least Recently Used)
    * Replaces the page which has not been referenced for the longest period of time
    * Recency
      * *Buffer pages which have not been referenced for a long time are less likely to be accessed sooner*
      * Temporal locality

3. Buffer Management

* Buffer-page replacement policy
  * LRU
    * Counter implementation
      * The buffer manager has a counter C
      * Whenever a page P is referenced
        * C is incremented
        * The new value for C is stored in P as well
      * The page with the lowest value for C is selected to be evicted
    * Disadvantage
      * The frequency and the scale of temporal locality may vary over time

3. Buffer Management

* Buffer-page replacement policy
  * LFU
    * Replaces the least frequency referenced buffer page
    * Frequency
      * *Buffer pages which have less frequently been referenced are less likely to be accessed sooner*

3. Buffer Management

* Buffer-page replacement policy
  * LFU
    * Counter implementation
      * A counter C is associated to each page in the buffer pool
      * Whenever a page P in the buffer pool is referenced
        * C is incremented
      * The page with the lowest value for C is selected to be evicted
* LRU x LFU Efficiency
  * Depends on the workload

3. Buffer Management

* Desafio (0,2 na 1a. Aval - 20 min)
  * Considere um mecanismo de gerenciamento de buffer com, onde a capacidade da região de buffer do SGBD é de 3 páginas. Suponha a execução de uma carga de trabalho que referencia cinco páginas distintas P_i, onde $0<i<6$. As páginas estão armazenadas inicialmente em memória secundária. O fluxo de páginas decorrente da execução desta carga é:
    * P_3 P_1 P_2 P_1 P_5 P_3 P_4 P_5 P_3 P_5 P_2 P_1 → Tempo
  * No fluxo acima, a primeira página referenciada é P_3, a segunda P_1 e assim por diante. Monte os mapas de alocação da região de buffer, utilizando as políticas LRU e LFU. Calcule ainda as taxas de acerto para as duas políticas
* Desafio (0,2 na 1a. Aval)
  * Questão 5 da 1a. lista de exercícios (até dia 18.03)

3. Buffer Management

* Buffer-page replacement policy
  * ARC
    * Dynamically adapts its behaviour
      * Recency x Frequency
    * Depending on workload features

3. Buffer Management

* Buffer-page replacement policy
  * ARC
    * Maintains LRU page lists: L_1 and L_2 (ghost buffers + buffer pages)
      * $|L_1| + |L_2| = 2c$
      * P_i is in L_1 if it has been referenced only once, since the last time P_i was evicted from $L_1 \cup L_2$
      * P_k is in L_2 if it has been seen at least twice, since the last time P_k was evicted from $L_1 \cup L_2$
    * L_1 captures recency and L_2 frequency
    * L_1 and L_2 behave as a cache directory holding 2c entries

3. Buffer Management

* Buffer-page replacement policy - ARC
    * L_1
        * T_1
            * Recent buffer entries
        * B_1
            * Entries (pages id) recently evicted from the T_1 cache, but are still tracked
    * L_2
        * T_2
            * Buffer pages referenced at least twice
        * B_2
            * Entries (pages id) recently evicted from the T_2, but are still tracked

3. Buffer Management

* Buffer-page replacement policy – ARC
  * Pages in T_1 and T_2 reside in the cache directory and in the page cache
  * Pages in B_1 and B_2 reside only in the cache directory
    * Maintain historical data used to predict hit rates in T1 and T2

3. Buffer Management

* Buffer-page replacement policy - ARC

4. Indexing

* Which is the most efficient procedure to find a given word in a book?
  * Go to the book's index
  * Search for the word in the index structure
    * The entries in the index structure are ordered
  * If there is an entry in the index equal to the word we are looking for
    * The entry has the address of the pages which contain the word
  * Otherwise, there is no occurrence for that word in the book

4. Indexing

* Each index entry has
  * A search key
    * One or more attributes
  * Pointers to pages which contain tuples with the value of the search key
* Reduces the amount of disk access for finding a subset of tuples
* Classification
  * Ordered
  * Non-ordered

4. Indexing
- Classification -

* Ordered index
  * Search keys are ordered in the index structure
  * Primary (Clustered)
    * The table is physically sorted by the search key
    * Dense
      * One entry for each tuple
    * Sparse
      * One entry for each data page
  * Secondary (Non-clustered)
    * The table is not physically sorted by the search key
    * Dense

4. Indexing
- Classification -

* Exercise
  * Why a secondary index can not be implemented as a sparse index ?
  * Differentiate the concepts of Primary Index and Primary Key

4. Indexing
- Classification -

* Non-ordered index
  * The search keys are not ordered in the index structure
  * Hash index

4. Indexing
- Hash Index -

* Hash Index
  * $h = \lfloor ID/3 \rfloor$

How many disk accesses are necessary to process Q ?

Q: Select * from Employee where ID=7

4. Indexing
- B+-Tree -

* B+-Tree properties
  * Search keys are ordered
  * Balanced tree
    * All paths from the root to the leaves have the same amount of nodes
    * Same length
* n-ary tree
  * There are at most n children per node
  * n represents the order of tree (fanout)
    * Branching factor

4. Indexing
- B^+-Tree -

* Rules for a B^+-Tree of order n (balancing)
  * The root has at least two pointers
  * Internal nodes have at most n pointers, which point recursively to a B^+-Tree
    * Internal nodes have at most n-1 search keys
    * Structure of an internal node
      * $<PT_1, K_1, PT_2, K_2, \ldots, PT_{m-1}, K_{m-1}, PT_m>$, where $m \leq n$
  * Internal nodes have at least $\lceil n/2 \rceil$ pointers
    * $\lfloor ((n-1)/2) \rfloor$ search keys

4. Indexing
- B+-Tree -

* Rules for a B+-Tree of order n (balancing)
  * Leaf nodes
    * At most n-1 and at least $\lfloor ((n-1)/2) \rfloor$ pointers
    * To data pages
  * The last pointer points to the next leaf node

4. Indexing
- B^+-Tree -

* Rules for a B^+-Tree of order n (order)
  * At each node
    * $K_1<K_2<K_3< \ldots <K_{m-1}$, where $m \leq n$
  * For every search-key value X belonging to subtree pointed by PT_i
    * For $i=1$, $X \leq K_1$
    * For $1<i<m$, $K_{i-1}<X \leq K_i$
    * For $i=m$, $X>K_{m-1}$

4. Indexing
- Inserting an Entry into aB+-Tree -

```
B+_Tree ( )
    Allocate a new node R; /* the root */
    While there are entries
        do
            insert_new_entry (L, K); /* L is the node address where to insert k */
     End-While;
insert_new_entry (L, K)
If L has n-1 search-keys        /* overflow */
|       Allocate new node L' sibling of L;  /* node split */
|       If Parent(L) ≠ ∅      /* L has a parent */
|           /* the median search-key value M should be inserted into Parent(L) */
|           insert_new_entry (Parent(L), M)
|       Otherwise
|           Allocate a new node R, where R is the parent of L and L´
|           /* the median search-key value M should be inserted into R */
|           insert_new_entry (R, M)
|           /* In R,  the pointer PT1 points to L and PT2 points to L' */
|       End-If
|       search-key values X ≤ M remain in node L;
|       search-key values Y > M should be inserted into L';
Otherwise
        allocate K in L
```

4. Indexing
- B+-Tree -

* Build a B+-Tree of order 3 with the following insertion sequence: 8,5,1,7,3,12,9,6

4. Indexing
- B+-Tree -

* Build a B+-Tree of order 3 with the following insertion sequence: 8,5,1,7,3,12,9,6

4. Indexing
- B^+-Tree -

* The height should be computed for the worst-case
  * Each node (including the root) contains $\lceil n/2 \rceil$ pointers
* Relation between a level L and the number (M) of nodes in L
  * $L=0;\ M=1$
  * $L=1;\ M=n/2$
  * $L=2;\ M=(n/2)^2$
  * $L=3;\ M=(n/2)^3$

    ⋮
  * $L=h;\ M=(n/2)^h;$
  * $\log_{n/2} M = \log_{n/2} (n/2)^h$

▶ $h = \log_{\lceil n/2 \rceil} M$ →

4. Indexing
- B+-Tree -

* Compute the height of a B+-Tree for the worst-case, where
  * The root has two pointers
  * Each node (except the root) contains $\lceil n/2 \rceil$ pointers
    * 0,5 ponto na avaliação da 1a. AP
      * 20 minutos

4. Indexing
- B+-Tree -

* Computing the order
  * Each node corresponds to a database page
    * Let T bytes be the size of a page, PT bytes be the size of a B+-Tree pointer and S bytes the size of a search key
    * Order of a B+-Tree
      * $T = n * PT + (n-1) * S$

4. Indexing

- Read/Write complexity in B+-Tree -

* Number of read pages to find a tuple with key K_2
  * $1 + \log_{n/2} M$
* Number of written pages to insert a new entry K'
  * $1 + \log_{n/2} M$
    * In worst-case, all nodes have n pointers
      * n corresponds to the B+-Tree order

4. Indexing
- B+-Tree -

* Deleting a node entry

4. Indexing
- Removing an entry in a B+-Tree -

Função remoção_entrada(L, chave_busca)
Se L é nó raiz
    remoção_entrada_raiz(L, chave_busca)
Senão
    remoção_entrada_nó(L, chave_busca)
Fim;

4. Indexing

- Removing an entry in a B+-Tree -

Função remoção_entrada_raiz(L, chave_busca)

Se L possui apenas uma entrada <chave,PD> /* L possui 2 filhos */
S=filho(L) /* retorna filho de L que possui no mínimo (n/2) entradas */
Se S ≠ ∅
Se S=filho_esquerda(L)
/* Inserir na raiz maior valor de chave de busca em S toda subárvore */
remoção_entrada(S, K)
inserção_nova_entrada(L, K)
Senão
/* Inserir na raiz menor valor de chave de busca em S */
remoção_entrada(S, K)
inserção_nova_entrada(L, K)
Senão /* fundir os nós filhos de L em um nó R que será a nova raiz da árvore */
Senão
remover entrada <Ki> de L
Se Pi ≠ NULL ou Pi+1 ≠ NULL
atualizar_chave busca(L,i)

4. Indexing

- Removing an entry in a B+-Tree -

Função remoção_entrada_nó(L, k)

```
remover entrada <K> de L;
Se nó L possui ((n- 1)/2) entradas <k>  /* underflow */
    L'= irmão (L);  /* retorna  irmão de L que possui (n/2) entradas <k> */
    Se L' ≠ ∅
      inserção_nova_entrada(L,k);   /*inserir V (valor entre L e L'  no pai(L)) em L */
      Se L'=irmão_esq(L) /* Substituir V no nó pai por maior valor de chave em L' */
        remoção_entrada(L', K); inserção_nova_entrada(Pai(L), K)
      Senão   /* Substituir V' no nó pai  por menor valor de chave de busca  em   L' */
        remoção_entrada(L', F); inserção_nova_entrada(Pai(L), F);
    Senão  /* Irmãos de L contêm exatamente n/2 - 1 entradas <k> */
     Se L'=irmão_esq(L)
      inserção_nova_entrada(L', V, PD); /*inserir V  em L' */
      remoção_entrada(Pai(L), V, PD); /*remove V de Pai(L) */
      inserção_nova_entrada(Pai(L), K); /*Substituir V em Pai(L) por maior valor K em L*
      Transferir entradas de L para L'; remover nó L;
     Senão
      inserção_nova_entrada(L, V, PD) /*inserir V  em L' */
      remoção_entrada(Pai(L), V, PD); /*remove V de Pai(L) */
      inserção_nova_entrada(Pai(L), K) /* Substituir V  no nó pai por maior chave em L' *
      Transferir entradas de L' para L; remover nó L';
Senão
    Se PTi ≠ NULL ou PTi+1 ≠ NULL
        atualiza_chave_busca(L,PTi);
```

4. Indexing

- Removing an entry in a B+-Tree -

Função atualiza_chave_busca(L, PTi)

S=filho(L, PTi) /*retorna verdade se filho de L apontado por PTI tem (n/2) entradas <chave,PD> */

S´= filho(L, PTi+1)

Se (S) /* Inserir na raiz maior valor de chave de busca da subárvore apontada PTI*/

remoção_entrada(L´, K)

inserção_nova_entrada(L, K, PD)

Senão

Se (S´) /* Inserir na raiz menor valor de chave da subárvore apontada PTi+1 */

remoção_entrada(L´, K)

inserção_nova_entrada(L, K, PD)

Senão

/* fundir os nós filhos de L em um nó S que será a nova raiz da árvore */

4. Indexing
- B+-Tree -

* Definindo-se índices no SQL Server

```
CREATE [UNIQUE] [CLUSTERED |
NONCLUSTERED] INDEX nome_índice
ON nome_tab (nome_col [ASC | DESC [,
{nome_col}])

INCLUDE (nome_col [, {nome_col}])

[WITH ([PAD_INDEX= ON | OFF]
FILLFACTOR = taxa_preenchimento_nós) ]
[ON filegroup]
```

4. Indexing
- B+-Tree -

* Cláusula INCLUDE
  * Permite o armazenamento de um ou mais atributos na estrutura de índice
    * Tais atributos não compõem a chave de busca do índice
  * Pode ser eficiente em consultas com projeções dos atributos especificados no include
    * Evita acesso adicional à tabela
  * Cuidado
    * Árvore pode ficar mais alta
    * Porque ?

4. Indexing
- B+-Tree -

* Taxa de preenchimento de nós
  * Fillfactor
    * Taxa de preenchimento de nós da árvore B+-Tree
      * Aplica-se na construção inicial da estrutura
    * [1,100] em percentual
  * Pad_Index
    * ON
      * Fillfactor aplica-se aos nós internos e nós folhas
    * OFF
      * Fillfactor aplica-se apenas aos nós folhas
  * Default
    * 0
      * pad_index=off, fillfactor=100

4. Indexing
- B+-Tree -

* Definindo-se índices no PostGres

CREATE [UNIQUE] INDEX [CONCURRENTLY] nome_índice ON nome_tab METHOD tipo_ind (nome_col [ASC | DESC [, {nome_col}])

[WITH ([FILLFACTOR = taxa_preenchimento_nós)] [TABLESPACE nome_tablespace]

* Tipo de índice
  * B+-Tree, hash, gist, spgist, gin

4. Indexing
- B+-Tree -

* Atividade lab I
  * Execute a seguinte consulta no TPC-H
    * (Q1) Select * from lineitem
      where l_orderkey=5572166
    * Marque o tempo de execução
  * Crie um índice secundário para lineitem em l_orderkey, com taxa de preenchimento de 100%, valendo para todos nós
    * Execute novamente a consulta e marque o tempo

4. Indexing
- B+-Tree -

* Atividade lab II
  * Remova todos os índices de lineitem
  * Crie um índice secundário para lineitem em l_orderkey, com taxa de preenchimento de 1%, valendo para todos nós
    * Marque o tempo de execução e compare com o da atividade I
  * Encerre a sessão com a consulta Q1
  * Inicie uma nova sessão e execute Q1 no TPC-H
    * (Q1) Select * from lineitem
      where l_orderkey=5572166
    * Observe o tempo de execução apresentado na Janela de Propriedades

4. Indexing
- B+-Tree -

* Atividade lab III
  * Especifique a seguinte consulta (índice secundário, com chave de busca l_orderkey)
    (Q2) Select l_orderkey, l_quantity from lineitem
    where l_orderkey=5572166
  * Visualize o plano de execução
  * Remova o índice (drop index)
  * Crie um índice secundário para lineitem em l_orderkey, com taxa de preenchimento de 100%, valendo para todos nós. Utilize a opção INCLUDE com a coluna l_quantity
  * Visualize novamente o plano de execução de Q2
  * Explique a diferença entre os planos de execução.

4. Indexing
- B+-Tree -

* Atividade lab IV
  * Especifique a seguinte consulta
    (Q3) Select * from lineitem
    where L_quantity>50
  * Visualize e explique o plano de execução

4. Indexing
- B+-Tree -

* Atividade lab V
  * Crie um índice secundário para lineitem em l_suppkey, com taxa de preenchimento de 100%, valendo só para nós folhas
  * Escreva Q4 e Q5. Verifique o plano de execução de cada uma

```
(Q4) select * from lineitem
     where l_suppkey>50 and l_suppkey<100
     order by l_suppkey
(Q5) select * from lineitem
     where l_suppkey>50 and l_suppkey<1000
     order by l_suppkey
```

  * Explique porque há diferenças entre os planos de execução

4. Indexing
- B+-Tree -

* Atividade lab VI
  * Remova todos os índices de lineitem.
  * Verifique o plano de execução de Q6

    (Q6) select * from lineitem
  * Crie um índice primário para lineitem em L_suppkey, com taxa de preenchimento máxima, valendo para todos nós.
  * Verifique o plano de execução de Q6
  * Explique a diferença entre os planos de execução

4. Indexing
- B+-Tree -

* Atividade lab VII
  * Remova todos os índices de lineitem.
  * Crie um índice primário para lineitem em l_orderkey e l_suppkey, com taxa de preenchimento máxima, valendo para todos nós.
  * Execute a seguinte consulta
    * (Q7) Select * from lineitem
      where l_orderkey=5700611 and
      l_suppkey=7641
    * (Q8) Select * from lineitem
      where l_orderkey=5700611
    * (Q9) Select * from lineitem
      where l_suppkey=7641
  * Explique a diferença entre os planos de execução

* Grid Files
  * Divisão de um espaço k-dimensional em células irregulares
    * Formando uma grade (grid) irregular
  * Estruturas básicas
    * k escalas lineares
      * Cada escala é definida pelas linhas que cortam o grid em uma dimensão
    * Diretório k-dimensional
      * Vetor k-dimensional
      * Cada célula no grid corresponde a uma entrada no diretório

* Grid Files

* Indexação por 2 chaves de busca (2D)

Diretório

y3	P_{11}	P_1	P_6	P_9
y2	P_3	P_7	P_8	P_{12}
y1	P_5	P_{10}	P_4	P_2
	x1	x2	x3	x4

- Como acessar dados de P(a,b)?
  - x2<a ≤ x3 e y1<b ç y2

* Grid Files

* Região de armazenamento

Diretório

P_3	P_3	P_6	P_6
P_3	P_3	P_8	P_8
P_5	P_5	P_5	P_2

6 3 2

800 980 1500 4500

* Grid Files

* Sobrecarga de página

Diretório

P_3	P_3	P_6	P_6	P_6
P_3	P_3	P_8	P_8	P_8
P_5	P_5	P_5	P_2	P_9

6, 3, 2

800 980 1500 3000 4500

Diretório

P_3	P_3	P_6	P_6
P_3	P_3	P_8	P_8
P_5	P_5	P_5	P_2

6, 3, 2

800 980 1500 4500

- ✳ Capacidade de cada página: 10 tuplas
- ✳ Inserir a tupla com salário=3900 e lotação=1

* Grid Files
  * Vantagens
    * Operações sobre pontos
    * Indexação através de múltiplos atributos
      * Busca mais eficiente que índice composto em árvores B^+
  * Desvantagem
    * Espaço para armazenar diretório
      * Crescimento exponencial com o crescimento de objetos indexados
    * Bucket overflow
    * Bucket underflow

* Quad-Trees
  * Divisão de um espaço k-dimensional em células irregulares
  * Árvore k-dimensional
    * Folhas apontam para um página
    * Várias folhas podem apontar para um mesmo bucket
    * Um nó em uma quad-tree k-dimensional
      * Representação de um ponto
      * Possui até 2^k filhos, onde cada filho
        * Raiz de uma subárvore correspondente a 2^k quadrantes

* Quad-Trees

* Um nó em uma quad-tree bidimensional
  * Possui até 2^2 filhos, onde cada filho
    * Raiz de uma subárvore correspondente a 2^2 quadrantes
    * Cada quadrante representa uma direção
      * NW, NE, SW, SE

* Quad-Trees

Salário
P2(35,80)
P5(70,60)
P1(55,45)
P3(10,35)
P6(60,25)
P4(20,15)
Lotação

P1
P5
P2
P3
P6
P4

* kd-Trees
  * Particionamento de um espaço k-dimensional em células irregulares
  * Árvore binária k-dimensional
  * Folhas apontam para duas páginas
  * Um nó na árvore
    * Representação de um ponto
      * Indica onde a divisão do sub-espaço foi realizada
  * Estrutura carregada por completo em memória principal

* kd-Trees
  * Discriminador
    * Valor associado a cada nível da árvore
    * Indica a dimensão D em que a divisão foi realizada
    * Um único valor por nível (árvore homogênea)
  * Cada nó P possui dois filhos
    * LOSON(P) - filho à esquerda
      * Valor do atributo referente a D de qualquer nó descendente de LOSON(P) é menor que o valor do mesmo atributo d P
    * HISON(P) - filho à direita
      * O valor do atributo referente a d de qualquer nó descendente de HISON(P) é maior que ou igual ao valor do mesmo atributo no nó P

* kd-Trees

Lotação
90
P3(25,80)
P2(10,60)
P7(80,55)
P4(20,50)
P1(60,40)
P8(15,30)
P5(75,25)
P6(70,10)
125
Salário
P1
P2
P5
P4
P3
P6
P7
P8

* kd-Trees
  * Estratégias de particionamento
    * Alternar a dimensão
    * Priorizar uma determinada dimensão
      * Consultas em extensão em uma dimensão específica
    * Particionamento em sub-espaços com distribuição uniforme de pontos
  * Estratégias para término do particionamento
    * Os pontos de cada célula podem ser armazenados em um único bucket
    * Determinar um número máximo de pontos por célula

* kd-Trees
  * Remoção de um nó interno P_k
    * Um outro nó deve ser definido para substituir P_k
    * Nó pertencente a sub-árvore que tem como raiz
      * Maior valor do filho à esquerda de Pk
      * Menor valor do filho à direita de Pk

* kd-Trees
  * Remoção de P5

* kd-Trees
  * Remoção de P5

* kd-Trees
  * Vantagens
    * Pode ser carregada na memória
    * Consultas em extensão
  * Desvantagens
    * Remoção de um nó
      * Reorganização da sub-árvore do nó removido
    * Árvore não balanceada
      * Pior caso
        * Lista
          * O(n)

5. Processamento de Consultas
- Conceitos Básicos -

* Consulta sobre BDs
  * Expressão de linguagem de consulta que descreve dados a serem acessados
* Linguagem de Consulta
  * Linguagem de alto nível
  * Além de consultar:
    * Definir a estrutura de dados (DDL componente)
    * Modificar dados (DML)
    * especificar restrições de integridade (e.g., not null)
    * especificar restrições de segurança

5. Processamento de Consultas
- Conceitos Básicos -

* Linguagens de Consultas (SGBDs Relacionais)
  * SQL (Structured Query Language)

```
SELECT d.nome
FROM Departamento d, Empregado e
WHERE d.cod_dep=e.cod_dep
```

  * Quel (Query Language)

```
RANGE OF (d,e) IS (Departamento, Empregado)
RETRIEVE INTO nome(d.nome)
WHERE (d.cod_dep=e.cod_dep)
```

  * QBE (Query By Example)

5. Processamento de Consultas
- Conceitos Básicos -

* Consultas em linguagem de alto nível
  * Mapeadas para outras formas de representação
    * Suporte para garantir transformações de consultas
      * Otimização
    * Expressar uma grande classe de consultas
* Formas de representação para BDs relacionais
  * Cálculo Relacional
  * Álgebra Relacional
  * Grafo de Consulta
  * Tableaus

5. Processamento de Consultas
- Conceitos Básicos -

* Álgebra Relacional
  * Coleção de operadores sobre relações
  * Linguagem procedimental
    * Descreve operações a serem executadas
  * Operações básicas
    * Seleção (σ)
    * Projeção (Π)

    } Operações unárias
    * União (U)
    * Diferença (-)
    * Produto cartesiano (x)

    } Operações binárias

5. Processamento de Consultas
- Conceitos Básicos -

* Operações derivadas
  * Junção natural (natural join)
    * $r \bowtie_{\theta_3} s = \Pi_{R \cup S} (\sigma_{r.A1\ \theta\ s.A1 \wedge r.A2\ \theta\ s.A2 \wedge \ldots \wedge r.An = s.An} (r \times s))$
      * where $r \cap s = \{A_1, A_2, \ldots, A_n\}$ and $\theta \in \{=, >, <, \leq, \geq, \neq\}$
  * Divisão
    * $r / s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$
  * Interseção
    * $r \cap s = r - (r - s)$

5. Processamento de Consultas
- Conceitos Básicos -

* Outer joins
  * Left outer join ⟕
  * Right outer join ⟖
  * Full outer join ⟗

* Consulta C:
  * Nome dos empregados que não possuem dependentes

$$\Pi_{nome_emp}(Empregado) - \Pi_{nome_emp}(Empregad \bowtie Dependente)$$

5. Processamento de Consultas
- Conceitos Básicos -

* Álgebra relacional
  * Define um algoritmo para a construção da relação resultado
  * Oportunidades de otimização podem estar escondidas em uma determinada sequência de operadores algébricos
  * "Relacionalmente" completa

5. Query Processing
- Phases -

5. Query Processing
- Query Optimization -

* Query Execution Plan (QEP)
  * Logical QEP
    * Algebraic operators
      * Relational Algebra
  * Physical QEP
    * Physical operators
      * Algorithms for implementing algebraic operators
        * For each logical operator
          * Several physical operators

5. Query Processing
- Query Optimization -

* Query Rewrite
  * Several relational algebra expressions for a given SQL query
  * Logical QEP
    * Operator graph
      * Directed Graph OG(Q)=(V,E)
        * v ∈ OG(Q) iff v is a relational algebra's operator in Q or v is a table in Q
        * An edge e ∈ E represents the data flow in Q

5. Query Processing
- Query Optimisation -

*select * from lineitem, partsupp, part, supplier*
where $\boxed{\text{l_partkey=ps_partkey and p_partkey=ps_partkey and ps_suppkey=s_suppkey}}$ *and* $\boxed{\text{l_quantity>9}}$

Join condition

Selection condition

5. Query Processing
- Query Optimization -

*select * from lineitem, partsupp, part, supplier*
where l_partkey=ps_partkey and
p_partkey=ps_partkey and
ps_suppkey=s_suppkey and l_quantity>9

Version 2
(Left deep tree)

5. Query Processing
- Query Optimization -

* Query Rewrite
  * Transformation rules
    * $\sigma_{\theta_1 \wedge \theta_2}(r) \Leftrightarrow \sigma_{\theta_1}(\sigma_{\theta_2}(r)) \Leftrightarrow \sigma_{\theta_2}(\sigma_{\theta_1}(r))$
    * $\Pi_{A_1}(\Pi_{A_2}(\Pi_{A_3} \ldots (\Pi_{A_n}(r))) \ldots) \Leftrightarrow \Pi_{A_1}(r)$
    * $\sigma_{\theta_1 \wedge \theta_2}(r \bowtie_{\theta_3} s) \Leftrightarrow \sigma_{\theta_1}(r) \bowtie_{\theta_3} \sigma_{\theta_2}(s)$
    * $\Pi_{A_1 \cup A_2}(r \bowtie_{\theta} s) \Leftrightarrow \Pi_{A_1 \cup A_2}((\Pi_{A_1 \cup A_3}(r) \bowtie_{\theta} (\Pi_{A_2 \cup A_4}(s))$
      * A_1 attributes of r
      * A_2 attributes of s
      * A_3 attributes of r in θ (join condition)
      * A_4 attributes of s in θ (join condition)

5. Query Processing
- Query Optimization -

* Query Rewrite
  * Heuristics
    * Push down selective operations
      * Selection and projection
        * Tuple and column filters
    * Build logical QEP as a left-deep tree
  * Output: Efficient Logical Plan

5. Query Processing
- Join Physical Operators -

* Atividade
  * Construa o plano lógico mais eficiente, para a seguinte consulta:

```
Select l_orderkey, s_name
from lineitem, partsupp, part, supplier
where l_partkey=ps_partkey and
p_partkey=ps_partkey and
ps_suppkey=s_suppkey and l_quantity>9
```

5. Query Processing
- Query Optimization -

select l_orderkey, s_name
from lineitem, partsupp, part, supplier
where l_partkey=ps_partkey and
p_partkey=ps_partkey and
ps_suppkey=s_suppkey and l_quantity>9

Version 1
(heuristic - selection downwards)

5. Query Processing
- Query Optimization -

select l_orderkey, s_name
from lineitem, partsupp, part, supplier
where l_partkey=ps_partkey and
p_partkey=ps_partkey and
ps_suppkey=s_suppkey and l_quantity>9

Version 2 (Most efficient logical plan) (heuristics - selection and projection downward and left-deep tree)

5. Query Processing
- Join Physical Operators -

* Atividade Lab I
  * Especifique a consulta
    (Q10) Select * from lineitem, partsupp, part, supplier where l_partkey=ps_partkey and p_partkey=ps_partkey and ps_suppkey=s_suppkey
  * Verifique no plano de execução
    * Vértices
      * Fatores physical/logical operation, estimated number of rows, estimated row size, predicate e output list
    * Arestas
      * fatores estimated row number, estimated row size

5. Query Processing
- Join Physical Operators -

* Atividade Lab II
  * Especifique a consulta

    (Q11) Select *
    from lineitem, partsupp, part, supplier
    where l_partkey=ps_partkey and
    p_partkey=ps_partkey and
    ps_suppkey=s_suppkey and l_quantity>9
  * Verifique no plano de execução os fatores para os vértices e arestas
    * Identifique e justifique a alteração com relação ao plano da Q10

5. Query Processing
- Join Physical Operators -

* Atividade Lab III
  * Especifique a consulta

    (Q12) Select l_partkey,ps_partkey,p_name,s_name
    from lineitem, partsupp, part, supplier
    where l_partkey=ps_partkey and
    p_partkey=ps_partkey and
    ps_suppkey=s_suppkey and l_quantity>9
  * Verifique no plano de execução os fatores para os vértices e arestas
    * Identifique e justifique a alteração com relação ao plano da Q11

5. Query Processing
- Query Optimization -

* Query Execution Plan Selection
  * Build and selection of QEP
    * Physical plan
  * Search space
    * Set of all possible physical plans
  * Criterion for selecting a physical plan
    * The lowest query execution cost
      * Execution cost
        * Well approximated by the number of disk I/O operations
          * Adopted metric to estimate query execution cost

5. Query Processing
- Query Optimization -

* Parameters for estimating query execution cost
  * n_R
    * cardinality of R
  * P_R
    * Size of R in pages
      * Amount of pages containing tuples of R
  * $V(a,R)$
    * Cardinality of $\Pi_a(R)$
    * If a is primary key in R
      * $V(a,R)$=???
  * f_R
    * Page factor
    * Number of tuples of R in a page

5. Query Processing
- Query Optimization -

* Parameters for estimating query execution cost
  * SF(a,R)
    * Selectivity factor of an attribute a in R
    * Number of tuples on average which satisfies a comparison condition on attribute a
    * Metric
      * Restrictiveness of the selection condition
        * Filter efficiency
      * High selectivity
        * Low number of tuples in the result
      * Low selectivity
        * High number of tuples in the result

5. Query Processing
- Query Optimization -

* Parameters for estimating query execution cost
  * SF(a,R)
    * Rules to compute SF(a,R)
      * Condition: a=K
        * SF= 1/V(a,R)
      * Condition: a>K (a<K)
        * SF= (K-Min(a,R))/(Max(a,R) - Min(a,R))*Size(R)
      * Condition: a=b
        * SF= 1/max{V(a,R),V(b,S)}
    * Histogram
      * Measures the frequency of occurrence for each distinct value of a given attribute in a table

5. Query Processing
- Query Optimization -

* Histograms SQL Server
  * Gerados
    * Para atributos que aparecem em consultas, como
      * Condição de operações de seleção
      * Condição de operações de junção
      * Argumentos em cláusulas group by
    * Para atributos, utilizados como chave de busca em estruturas de índices

5. Query Processing
- Query Optimization -

* Histograms SQL Server
  * Steps
    * Quantidade de intervalos de valores para o atributo
      * Máximo de 200
    * Criados pelo SQL Server
  * Rows Sampled (RS)
    * Número de tuplas utilizadas nas estatísticas
    * Se RS < Cardinalidade
      * Resultados apresentados no histograma e densidade são calculados com base no RS
  * Range Hi-Key
    * Limite superior para um intervalo

5. Query Processing
- Query Optimization -

* Histograms SQL Server
  * Density
    * Definido por 1/DV
      * DV representa o número de valores distintos do atributo, excluindo-se os valoes de Range-Hi-Key
    * Indica a taxa de valores duplicados
      * Quanto menor DV, maior a densidade de valores repetidos
        * Seletividade baixa
    * Utilizado até o SQL Server 2008
      * No caso de um conjunto de atributos um único valor
  * Range Rows
    * Número de tuplas no intervalo, excluindo tuplas com valores do Range-Hi-Key

5. Query Processing
- Query Optimization -

* Histograms SQL Server
  * Distinct Range Rows
    * Número de tuplas com valores distintos para o atributo no intervalo, excluindo-se tuplas com valores do Range-H-Key
  * AVG Range Rows
    * Número de tuplas com valores repetidos para o atributo no intervalo, excluindo tuplas com valores do Range-H-Key
    * AVG=Range_Rows/Distinct_Range_Rows
    * Se distinct_range_rows = 0
      * AVG_range_rows = 1

* Histogram for estimating SF(a,R)

Table Name: dbo.lineitem

Statistics Name: _WA_Sys_00000001_7B905C75

Statistics for INDEX '_WA_Sys_00000001_7B905C75'.

Name	Updated	Rows
_WA_Sys_00000001_7B905C75	Apr 3 2019 12:09PM	6001215

All Density	Average Length	Columns
2.667591E-05	4	l_orderkey

Histogram Steps

RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS
68	0	127.7049
64036	64359.68	314.8813
70947	17840.05	314.8813
90726	25789.04	314.8813
118595	28137.6	314.8813
179905	34596.15	314.8813
206438	18562.69	314.8813
221543	19420.82	314.8813
242886	22175.86	314.8813
272738	20143.45	314.8813
326625	61785.3	314.8813
359111	28724.74	314.8813
399683	46158.31	314.8813
471811	53700.82	314.8813
488708	23395.31	314.8813
499585	17298.08	314.8813
516449	20866.09	314.8813
533665	19240.16	314.8813
596487	55642.9	314.8813
681350	66934.07	314.8813
701508	18246.54	314.8813

cardinalidade da tabela

206438 < l_orderkey ≤ 221543

206438 < l_orderkey < 221543

l_orderkey = 221543

5. Query Processing
- Query Optimization -

* Histogram for estimating SF(a,R)

Table Name: dbo.lineitem

Statistics Name: _WA_Sys_00000001_7B905C75

Rows Sampled	Steps	Density
132901	169	0.275165

Quantidade de valores distintos em (206438, 221543)

Quantidade média de tuplas com valores repetidos em (206438, 221543)

AVG=Range_Rows/Distinct_Range_Rows

DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
0	1
410	156.9802
117	152.2729
171	151.1785
182	154.6968
218	158.8072
126	146.8502
124	156.1778
147	150.9025
126	159.3556
401	154.1785
187	153.5868
295	156.5167
343	156.4706
153	152.791
122	141.4446
131	159.8744
128	149.7748
360	154.7183
448	149.4057
128	142.04

5. Query Processing
- Query Optimization -

* Atividade lab IV
  * Remova todos os índices de lineitem
  * Especifique as seguintes consultas
    (Q12) Select * from lineitem
    where l_orderkey=221543
    (Q13) Select * from lineitem
    where l_orderkey=231555
    (Q14) Select * from lineitem
    where l_orderkey<326683
  * Explique como é estimado fator de seletividade para as consultas acima
    * Consulte o plano estimado de execução e o histograma para l_orderkey

5. Query Processing
- Query Optimization -

* Vetor de densidade (SQL Server)
  * All Density
    * Por atributo
    * Por conjunto de atributos
      * Definidos como chave para um determinado índice
  * Definido por 1/DV
    * DV representa o número de valores distintos para cada prefixo do conjunto de atributos
    * Indica a taxa de valores duplicados
    * Quanto menor DV, maior a densidade de valores repetidos

5. Query Processing
- Query Optimization -

* Vetor de densidade (SQL Server)
  * Exemplo
    * Remover os indices de lineitem
    * Criar índice secundário I1 com chave de busca composta:
      * L_partkey
      * L_suppkey
      * L_orderkey
    * Executar o comando
      * dbcc show_statistics (lineitem, I1) with density_vector

* Vetor de densidade (SQL Server)

SQLQuery1.sql - BRAYNER-VM\BRAYNER.tpc_h (Brayner-VM\Brayner (54))* - Microsoft SQL Server Management Studio

```
dbcc show_statistics (lineitem, I1)
with density_vector
```

	All density	Average Length	Columns
1	5E-06	4	l_partkey
2	1.250718E-06	8	l_partkey, l_suppkey
3	1.666332E-07	12	l_partkey, l_suppkey, l_orderkey

Query executed successfully. | BRAYNER-VM\BRAYNER (12.0 RTM) | Brayner-VM\Brayner (54) | tpc_h | 00:00:00 | 3 rows

5. Query Processing
- Query Optimization -

* Physical Query Execution Plan
  * Algebraic operators in logical plans are substituted by physical operators
* Selection
  * Logical operator
    * Find tuples in a table R which satisfies the selection condition (R.aθc)
      * $\sigma_{R.a\theta c}(R)$
  * Physical operators
    * Implementation of the algebraic operator
      * Table Scan
      * Index Seek
      * Index Scan

5. Query Processing
- Selection Physical Operators -

* Table Scan on table R ($\sigma_{R.a\theta c}$ (R))
  * All the tuples belonging to table should be read
    * To check if R.aθc holds
  * Algorithm

TableScan ($\sigma_{R.a\theta c}$ (R))

Result = ∅
For each tuple $t_R \in R$ do
If tr.a = c
Result = Result ∪ $\{t_R\}$
End-For
End

  * Estimated cost
    * $EC = P_R$

5. Query Processing
- Selection Physical Operators -

* Implementação da seleção utilizando índice
  * Dois operadores físicos
    * Index Seek
    * Index Scan
  * Para cada operador físico
    * Há variações de implementação em função da configuração física do índice
      * Tipo de índice
        * Ordenado
          * Primário ou secundário
        * Hash
      * Chave de busca
        * Chave primária ou não
      * Estrutura da tabela (índice primário)
        * Árvore B^+ ou arquivo heap

5. Query Processing
- Selection Physical Operators -

* Index Seek
  * Primary Index (Clustered) i on R.a
    * R.a is primary key
      * R as B^+-tree
        * Leaf nodes are data pages
        * Algorithm

IndexSeek_PI_PK_BS ($\sigma_{R.a\theta c}$ (R))

Result = ∅
K=c / K is the search key */*
Search in B^+-tree a leaf-node LN for K
For each tuple $t_R \in$ LN do
If $t_R.a$= K
Result = Result ∪ {t_R}
Break
End-For
End

* Estimated cost
  * EC= HT_i
    * HT_i is the height of i

5. Query Processing
- Selection Physical Operators -

* Index Seek
  * Primary Index (Clustered) i on R.a
    * R.a is primary key
      * R as a heap file
        * Algorithm

IndexSeek_PI_PK_HF ($\sigma_{R.a\theta c}$ (R))

Result = ∅
K=c / K is the search key */*
*Search in B⁺-tree's leaf-node entry with page address **PT** to find a tuple with $t_r.a=K$*
Read Page(PT)
For each tuple $t_R \in$ page(PT) do
If $t_R.a= K$
Result = Result ∪ $\{t_R\}$
Break
End-For
End

For a search-key value K belonging to a data page pointed by PT_i holds:
For i=1, $K \leq K_1$
For $1<i<m$, $K_{i-1}<K \leq K_i$

* Estimated cost
  * $EC= HT_i + 1$

5. Query Processing
- Selection Physical Operators -

* Index Seek
  * Primary Index i on R.a
    * R.a is not primary key
      * R as B^+-tree
        * Several leaf nodes might be read
        * Algorithm

IndexSeek_PI_NPK_BS ($\sigma_{R.a\theta c}$ (R))

Result = ∅
K=c / K is the search key */*
Search in B^+-tree leaf-nodes LN_i for K
For 0<i≤X
  For each tuple $t_R \in LN_i$ do
    If $t_R.a = K$
      Result = Result ∪ $\{t_R\}$
  End-For
  i=i+1
End-For
End

tuplas que satisfazem condição de seleção estão agrupadas em nós contíguos

$X = \lceil SF(a,R)/f_R \rceil$

primeiro nó folha com tuplas que satisfazem a condição de seleção, já encontrado ao descer a altura da árvore

* Estimated cost
  * $EC = HT_i + (\lceil SF(a,R)/f_R \rceil) - 1$

5. Query Processing
- Selection Physical Operators -

* Index Seek
  * Primary Index i on R.a
    * R.a is not primary key
      * R as heap file
        * Several data pages might be read (file)
        * Algorithm

IndexSeek_PI_NPK_HF ($\sigma_{R.a\theta c}$ (R))

Result = ∅
K=c / K is the search key */*
*Search in B⁺-tree's leaf-node entry with page address **PT** to find a tuple with $t_r.a=K$*
Read Page(PT)
For $0<i\leq X-1$ ← $X=\lceil SF(a,R)/f_R \rceil$
  For each tuple $t_R \in$ Page do
    If $t_R.a= K$
      Result = Result ∪ $\{t_R\}$
  End-For
  i=i+1
Read NextPage
End-For
End

* Estimated cost
  * $EC= HT_i + (\lceil SF(a,R)/f_R \rceil)$

5. Query Processing
- Selection Physical Operators -

* Index Seek
  * Secondary index (non-clustered) i on R.a
    * R.a is primary key
      * Algorithm
        * Same as IndexSeek_PI_PK_HF ($\sigma_{R.a\theta c}$ (R))
        * A data page should be accessed
      * Estimated cost
        * $EC = HT_i + 1$

5. Query Processing
- Selection Physical Operators -

* Index Seek
  * Secondary index (non-clustered) i on R.a
    * R.a is not primary key
      * Algorithm
        * Simila to IndexSeek_PI_NPK_HF ($\sigma_{R.a\theta c}$ (R))
          * Except for the value of X
            * Worst case
              * Each tuple is located in a different page
                * X= SF(a,R)
      * Estimated cost
        * EC= HT_i + SF(a,R)

5. Query Processing
- Selection Physical Operators -

* Index Seek
  * Hash Index
    * Hash function on R.a
    * Estimated cost
      * EC= 2
        * Without bucket overflow (best case)
      * EC= 2 + n
        * n represents the list size
          * With bucket overflow (worst case)

5. Query Processing
- Selection Physical Operators -

* Index Scan
  * All leaf nodes should be visited
    * Let M be the number of leaf nodes and n the order of i
  * Primary Index
    * Estimated cost
      * R as B^+-tree
        * $EC = HT_i + M - 1$
      * R as a heap file
        * For each leaf node LF, for each pointer, the data page should be visited
        * $EC = HT_i + (M-1) + (n-1).M$

5. Query Processing
- Selection Physical Operators -

* Index Scan
  * Secondary Index
    * Estimated cost
      * $EC = HT_i + (M-1) + (n-1).M$

5. Query Processing
- Selection Physical Operators -

* Atividade lab V
  * Remova todos os índices de lineitem
  * Crie um índice secundário para lineitem em l_orderkey, com taxa de preenchimento de 100%, valendo para todos nós. Utilize a opção INCLUDE com a coluna l_quantity
  * Especifique a seguinte consulta (Q15) Select * from lineitem
    where l_quantity>50
  * Visualize e explique o plano de execução

5. Query Processing
- Selection Physical Operators -

* Atividade lab VI
  * Remova todos os índices de lineitem
  * Crie um índice secundário para lineitem em l_suppkey, com taxa de preenchimento de 100%, valendo para todos nós.
  * Especifique a seguinte consulta
    (Q16) Select * from lineitem
    where l_suppkey=50
  * Visualize e explique o plano de execução de Q16
  * Crie um índice primário para lineitem em l_orderkey, com taxa de preenchimento de 100%, valendo para todos nós
    * Visualize e explique o plano de execução para Q16

5. Query Processing
- Selection Physical Operators -

* Atividade lab VII
  * Remova todos os índices de lineitem
  * Crie um índice primário para lineitem em l_orderkey e l_suppkey, com taxa de preenchimento máxima, valendo para todos nós
  * Execute as seguintes consultas

    (Q17) Select * from lineitem
    where l_orderkey=5700611

    (Q18) Select * from lineitem
    where l_orderkey=0

    (Q19) Select * from lineitem
    where l_suppkey=7641

    (Q20) Select * from lineitem
    where l_suppkey=1
  * Explique as diferenças entre os planos de execução e justifique o que provoca tais diferenças

5. Query Processing
- Join Physical Operators -

* Join
  * Logical operator
    * Associate tuples of two tables through the join condition ($R \bowtie_{R.a=S.b} S$)
  * Physical operators (most implemented by commercial database systems)
    * Nested-loop Join
    * Index Nested-loop Join
    * Block Nested-loop Join
    * Merge Join
    * Hash Join

5. Query Processing
- Join Physical Operators -

Nested-loop Join ($R \bowtie_{R.a=S.b} S$)

For each tuple $t_r \in R$ do
  For each tuple $t_s \in S$ do
    If $t_r.a = t_s.b$
      Result = Result $\cup$ $\{(t_r,t_s)\}$
  End-For
End-For

$$EC = P_R + (n_r * P_S)$$

Size of R
- **P_R pages**
- **n_r tuples**

Size of S
- **P_s pages**
- **n_s tuples**

5. Query Processing
- Join Physical Operators -

Block Nested-loop Join ($R \bowtie_{R.a=S.b} S$)

For each Page P_r of R do
  For each Page P_S of S do
    For each tuple $t_s \in P_S$ do
      For each tuple $t_r \in P_r$
        If $t_r.a = t_s.b$
          Result = Result $\cup$ $\{(t_r,t_s)\}$
      End-For
    End-For
  End-For
End-For

$$EC = P_R + (P_R * P_S)$$

5. Query Processing
- Join Physical Operators -

* Atividade Lab VIII
  * Remova todos os índices das tabelas Part e Partsupp
  * Defina a consulta

    (Q21) Select ps_partkey, p_partkey, p_name
    from part inner join partsupp
    on ps_partkey=p_partkey
    where ps_partkey = 20
  * Visualize e explique o plano de execução gerado

5. Query Processing
- Join Physical Operators -

Index Nested-loop Join ($R \bowtie_{R.a=S.b} S$)

- An index is defined for R on R.a

For each tuple ts ∈ S do
  If (Index_Seek(Ind(R), t_s.b))==True
    Result = Result ∪ {(t_r,t_s)}
End-For

$EC = P_S + (n_s * C)$,
where C represents the cost of an Index Seek

5. Query Processing
- Join Physical Operators -

* Atividade Lab IX
  * Crie um índice secundário em Partsupp, tendo como chave de busca ps_partkey
  * Defina a consulta

```
(Q22) Select ps_partkey, p_partkey, p_name
        from part inner join partsupp
      on ps_partkey=p_partkey
      where ps_partkey = 20
```

  * Visualize e explique o plano de execução gerado

5. Query Processing
- Join Physical Operators -

Merge Join ($R \bowtie_{R.a=S.b} S$)

- The tables should be sorted by R.a and S.b
- For some DBS
  - There can not exist repetition on the attribute of the join condition in at least one table (external table)

Read R
For each tuple $t_s \in S$ do
  While $t_s.b \geq t_r.a$
    If $t_r.a = t_s.b$
      Result = Result $\cup$ $\{(t_r,t_s)\}$
    End-If
    Read R
  End-While
End-For

$EC = P_R + P_S$

5. Query Processing
- Join Physical Operators -

* Atividade Lab X
  * Cenário 1
    * Remova todos os índices de lineitem e part
  * Defina a consulta Q23 e guarde o plano de execução estimado

```
(Q23) Select *
    from part inner join lineitem
    on p_partkey=l_partkey
```

5. Query Processing
- Join Physical Operators -

* Atividade Lab X
  * Cenário 2
    * Crie um índice primário em Part, tendo como chave de busca p_partkey, especificando p_partkey como UNIQUE
    * Crie um índice primário em lineitem, tendo como chave de busca l_partkey
  * Observe plano de execução gerado para o Cenário 2
    * Explique e justifique a diferença com o plano de execução do Cenário 1

5. Query Processing
- Join Physical Operators -

* Atividade Lab X
  * Cenário 3
    * Remova o índice de part
    * Crie um índice secundário em Part, tendo como chave de busca p_partkey, especificando p_partkey como UNIQUE
    * Visualize o novo plano de execução estimado para a consulta Q23
      * Justifique a diferença com o plano do cenário 2 (com índice primário em part)

5. Query Processing
- Join Physical Operators -

* Atividade Lab XI
  * Especifique a consulta Q24
    * (Q24) Select l_orderkey, p_partkey, l_quantity, l_suppkey
      from part inner join lineitem
      on p_partkey=l_partkey
  * Observe que a diferença entre Q23 e Q24 refere-se apenas à operação de projeção
    * Justifique a diferença com o plano do cenário 3 da atividade X (slide anterior)

5. Query Processing
- Join Physical Operators -

* Atividade Lab XII
  * Remova todos os índices de lineitem e part
  * Crie um índice secundário em lineitem, tendo como chave de busca l_partkey
  * Crie um índice primário em Part, tendo como chave de busca p_partkey, especificando p_partkey como UNIQUE
  * Observe agora o plano de execução de Q25
    * (Q25) Select * from part inner join lineitem on p_partkey=l_partkey
  * Escreva a consulta Q26
    * (Q26) Select p_partkey, p_name, p_type, l_partkey
      from part inner join lineitem on
      p_partkey=l_partkey
  * Observe o plano de execução estimado para Q26
    * Justifique a diferença com o plano de Q25

5. Query Processing
- Join Physical Operators -

* Grace Hash Join ($R \bowtie_{R.a=S.b} S$)
  * 1st Phase (build phase)
    * R and S are physically partitioned by applying a hash function f on join attributes (R.a and S.b)

For each tuple $t_r \in R$ do
$i = \mathbf{f}(t_{r.a})$
$PR_i = PR_i \cup \{t_r\}$

For each tuple $t_s \in S$ do
$i = \mathbf{f}((t_{s.b})$
$PS_i = PS_i \cup \{t_s\}$

$$EC^B = 2(P_R + P_S)$$

5. Query Processing
- Join Physical Operators -

* Grace Hash Join ($R \bowtie_{R.a=S.b} S$)
  * 2nd Phase (probe phase)
    * Index nested-loop join on each pair of partition with same hash address
      * Hash index

For each partition $PR_k \in R$
For each tuple $t_r \in PR_k$ do
*i=**h**($t_{r.a}$)*
$B_i = B_i \cup \{(t_{r.a}, Pointer)\}$
end;
end;

For each tuple $t_s \in PS_k$ do
*i=**h**($t_{s.b}$)*
case $t_{s.b} \in B_i$
Result = Result $\cup \{(t_r, t_s)\}$
end;
end;

$$EC^P = P_R + P_S$$

5. Query Processing
- Join Physical Operators -

* Grace Hash Join ($R \bowtie S$)
  * Total estimated cost
    * $EC = 2(P_R + P_S) + (P_R + P_S)$
      * $EC = 3(P_R + P_S)$

5. Query Processing
- Join Physical Operators -

* In-memory Hash Join ($R \bowtie S$)
  * Build table (smallest table)
    * P_b
  * Probe table
    * P_p
  * In-memory hash table

* In-memory Hash Join ($R \bowtie S$)

for each row t_b in the build table
begin
calculate hash value on t_b join key // $t_b.jk_b$
insert t_b into the appropriate hash bucket // In-memory bucket
end
for each row t_p in the probe table
begin
calculate hash value on t_p join key // $t_p.jk_p$
for each row t_b in the corresponding hash bucket
if t_b joins with t_p // $t_b.jk_b = t_p.jk_p$
Result = Result $\cup$ $\{(t_b,t_p)\}$
end

5. Query Processing
- Join Physical Operators -

* In-memory (classic) Hash Join ($R \bowtie S$)
  * Estimated Cost
    * The build table is read several times
      * Probe table's cardinality (n_p)
    * $EC = P_b + P_p + n_p$

      # Read build table's pages (every tuple of probe table satisfies join condition)
    * Worst case
      * There is not enough main memory for holding the hash table
* SQL Server, Oracle, Postgres

5. Query Processing
- Join Physical Operators -

* Atividade Lab XIII
  * Cenário 1
    * Remova todos os indices das tabelas
      * lineitem, partsupp, part, supplier
  * Especifique a consulta
    * (Q27) Select l_partkey, ps_partkey, p_name, s_name
      from lineitem, partsupp, part, supplier
      where l_partkey=ps_partkey and
      p_partkey=ps_partkey and
      ps_suppkey=s_suppkey and l_quantity>9
  * Verifique e guarde o plano de execução de Q27

5. Query Processing
- Join Physical Operators -

* Atividade Lab XIII
  * Cenário 2
    * Crie um índice primário na tabela supplier, com chave de busca s_suppkey
    * Crie um índice secundário em partsupp, com chave de busca ps_suppkey. Utilize a opção INCLUDE com a coluna ps_partkey
    * Crie um índice secundário em lineitem, com chave de busca l_partkey. Utilize a opção INCLUDE com a coluna l_quantity
  * Analise o plano de execução de Q27 para o Cenário 2
    * Justifique a diferença com o plano do cenário Cenário 1 (sem índices)

5. Query Processing
- Query Optimization -

* Each physical operator has a different cost
  * Selection
    * Table Scan
    * Index Seek
    * Index Scan
  * Join
    * Nested-loop join: $P_r + n_r \cdot P_s$
    * Index nested-loop join: $P_r + n_r \cdot C$
    * Block nested-loop join: $P_r + P_r \cdot P_s$
    * Merge join: $P_r + P_s$
    * Hash join: $3(P_r + P_s)$

5. Query Processing
- Query Optimization -

* Query Execution Plan Selection
  * Join
    * Binary operation
      * Two operands (tables)

*select * from lineitem l, partsupp ps, part p, supplier s where l_partkey=ps_partkey and p_partkey=ps_partkey and ps_suppkey=s_suppkey*

    * Possible execution orders
      * $(((l \bowtie ps) \bowtie p) \bowtie s)$
      * $(((l \bowtie s) \bowtie p) \bowtie ps)$
      * $(((s \bowtie ps) \bowtie l) \bowtie p)$
      * $(l \bowtie ps) \bowtie (p \bowtie s)$ ...
      * 24 different execution orders
        * Search space

5. Query Processing
- Query Optimization -

* Query Execution Plan Selection
  * Join ordering problem
    * n-way joins executed as a sequence of 2-way joins
    * Search space
      * All possible execution orders
    * Search space size
      * n!
    * Complexity to find the most efficient execution order
      * O(n!)

5. Query Processing
- Query Optimization -

* Query Execution Plan Selection
  * Join ordering problem
    * n-way joins executed as a sequence of 2-way joins
    * Search space
      * All possible execution orders
    * Search space size
      * n!
    * Complexity to find the most efficient execution order
      * O(n!)

5. Query Processing
- Query Optimization -

* Query Execution Plan Selection
  * Join ordering problem
    * Search space inflating
      * There are four different physical operators for join algebraic operator
    * Search space deflating
      * Heuristic - left deep tree
        * Search space
          * (2n)!/(n+1)!n!
            * Desafio: Demonstração - 0,5 ponto 2a. Aval

5. Query Processing
- Query Optimization -

* Query Execution Plan Selection
  * Join ordering problem
    * Dynamic programming
      * $O(n \cdot 2^{(n-1)})$
    * Heuristics
      * Left deep tree
      * Join selectivity

5. Query Processing
- Join Physical Operators -

* Atividade Lab XIV (Join enumeration)
  * Visualize o plano de execução estimado para as seguintes consultas. Identifique e justifique as diferenças

(Q28)
```
Select l_orderkey, s_name
from lineitem, partsupp, part, supplier
where l_partkey=ps_partkey and
p_partkey=ps_partkey and
ps_suppkey=s_suppkey and l_quantity>9
```

(Q29)
```
Select l_orderkey, s_name
from lineitem, partsupp, part, supplier
where l_partkey=ps_partkey and
p_partkey=ps_partkey and
ps_suppkey=s_suppkey and l_quantity>50
```

5. Query Processing
- Query Physical Plan [Left Deep Tree] -

SQLQuery1.sql - BR...r-VM\Brayner (54))*

```
select * from lineitem, partsupp, part, supplier
where l_partkey=ps_partkey and p_partkey=ps_partkey and ps_suppkey=s_suppkey
and   l_quantity>9
```

100 %

Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
select * from lineitem, partsupp, part, supplier where l_partkey=ps_partkey and p_partkey=ps_part
Missing Index (Impact 12.7387): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON
```

5. Query Processing
- Query Optimization -

* Execution of a Query Execution Plan
  * Temporary (internal) results
    * Materialization
      * Stored
        * Secondary memory
    * Pipelining
      * Results are consumed by the next operator as soon as they are yield
  * Non Stop-and-go physical operators

6. Processamento de Transações
- O Problema de Concorrência em DBS -

* Banco de Dados
  * Estado
  * Estado consistente
    * Restrições de integridade são satisfeitas
  * Transição de estado

6. Processamento de Transações
- O Problema de Concorrência em SBDs -

* Concorrência em um ambiente multiusuário
  * Entrelaçamento de operações
    * Operações de um programa podem ser executadas entre duas operações de outro programa
  * Alterações inconsistentes no banco de dados
    * Transições gerando estados inconsistentes do BD
      * Transições incorretas

* Transições Incorretas

Tempo

qtde = 100

```
begin transaction T1
declare @valor dec(9,2)
select @valor=qtde from Produtos
        where id_produto=1
set @valor=@valor+20
```

```
begin transaction T2
declare @valor dec(9,2)

select @valor=qtde from Produtos
        where id_produto=1
set @valor=@valor-5
update Produtos set qtde=@valor
        where id_produto=1
commit
```

```
update Produtos set qtde=@valor
        where id_produto=1
commit
```

qtde = 120

Transição de estado incorreta

6. Processamento de Transações
- O Problema de Concorrência em SBDs -

* Entrelaçamento de operações de programas distintos
  * Pode provocar inconsistências no BD
* SGBD precisa monitorar e controlar
  * Execução concorrente de operações sobre o banco de dados
* Controle de Concorrência
  * Escalonador (Scheduler)

6. Processamento de Transações
- O Problema de Concorrência em SBDs -

* Transação (Eswaram et al)
  * Abstração que representa a seqüência de operações sobre bancos de dados resultantes da execução de um programa
  * Seqüência de *reads (r)* e *writes (w)*

```
begin transaction T2
declare @valor dec(9,2)
select @valor=qtde from Produtos
        where id_produto=1
set @valor=@valor-5
update Produtos set qtde=@valor
          where id_produto=1
commit
```

* $T_2=r_2(id_produto=1)w_2(id_produto=1)$

6. Processamento de Transações
- O Problema de Concorrência em SBDs -

* Execução concorrente de transações
  * Entrelaçamento
  * Transição correta ⇔ Entrelaçamento correto
  * Quando um entrelaçamento está correto ??
    * Produz um estado consistente
  * Impraticável
    * Muitas restrições de integridade estão especificadas nos programas e não no BD
    * Problema NP-difícil
* Verificar a execução (entrelaçamento) para determinar a corretude

6. Processamento de Transações

- O Problema de Concorrência em SBDs -

Escalonamento ou história
(schedule or history)

Controlar concorrência
⇔
sincronizar a execução de operações de transações concorrentes para produzir um estado consistente do banco de dados

6. Processamento de Transações

- Execução Correta de Transações -

* Premissa
  * A execução de uma transação é correta, se executada isoladamente
  * Produz sempre um estado consistente
* Teorema
  * Toda **execução serial** de transações é correta
  * $S = T_2 T_5 T_1 T_4 T_3 \ldots T_{n-1} T_n$
  * **Padrão para corretude** de schedules
* Necessidade de um critério que garanta mais paralelismo

6. Processamento de Transações
- Execução Correta de Transações -

* Conjectura
  * Seja um schedule S sobre $\mathcal{T}=\{T_1, T_2, ..., T_n\}$
    * Se S produz um estado do banco de dados igual ao produzido por alguma execução serial do conjunto de transações $\mathcal{T}$
      * A execução de S é correta
* Serializabilidade
  * Ilusão que transações são executadas de forma isoladas
    * Sem interferências de outras transações

6. Processamento de Transações
- Execução Correta de Transações -

* Equivalência de schedules
  * Dado um conjunto $\mathcal{I}=\{T_1, T_2, ..., T_n\}$ de transações
    * Encontrar escalonamentos, cujas execuções produzam o mesmo estado no BD que a execução de algum schedule serial sobre $\mathcal{I}$
* Equivalências
  * Estado final
  * Visão
  * Conflito

6. Processamento de Transações
- Execução Correta de Transações -

* Equivalência de estado final
  * Dois Schedules S e S' sobre $\mathcal{T}=\{T_1, T_2, ..., T_n\}$ são equivalentes de estado final ($S \approx_{FS} S'$) se, e somente se,

    (i) têm as mesmas operações pertencentes às transações de $\mathcal{T}$ e

    (ii) produzem o mesmo estado final no BD, se forem executados sobre um mesmo estado inicial
  * Schedules não capturam o conceito de estados inicial e final

6. Processamento de Transações
- Execução Correta de Transações -

* Equivalência de visão
  * Dois Schedules S e S' sobre $\mathfrak{I}=\{T_1, T_2, ..., T_n\}$ são equivalentes de visão ($S \approx_V S'$) se, e somente se,

    (i) têm as mesmas operações pertencentes a transações de $\mathfrak{I}$;

    (ii) para cada operação $r_i(x)$ de $T_i \in \mathfrak{I}$, o valor lido por $r_i(x)$ é o mesmo nos dois schedules e;

    (iii) Se $w_i(x)$ é a última operação de escrita sobre x em S, então $w_i(x)$ é a última operação de escrita sobre x em S'
* Controle sobre todas operações de leitura e escrita

6. Processamento de Transações

- Execução Correta de Transações -

* Operações em conflito
  * Operações de transações distintas
  * Executadas sobre um mesmo objeto e
  * Pelo menos uma delas é uma operação de escrita
* Exemplo
  * $T_1 = r_1(x) w_1(x)$; $T_2 = r_2(x) w_2(y)$
  * $S = r_2(x) r_1(x) w_2(y) w_1(x)$
* Relação de precedência
  * Em S, $r_2(x) <_S w_1(x)$
    * $r_2(x)$ é escalonada antes de $w_1(x)$

6. Processamento de Transações
- Execução Correta de Transações -

* Equivalência de conflito
  * Dois Schedules S e S' sobre $\mathcal{T}=\{T_1, T_2, ..., T_n\}$ são equivalentes de conflito ($S \approx_C S'$) se, e somente se,

    (i) têm as mesmas operações de transações em $\mathcal{T}$ e

    (ii) Ordenam operações em conflito da mesma forma
    * $\forall p_i \in OP(T_i), q_j \in OP(T_j), i \neq j$, p_i e q_j em conflito,
      * Se $p_i <_S q_j$, então $p_i <_{S'} q_j$
  * Exemplo
    * $T_1=r_1(y)r_1(x)w_1(y)$; $T_2=r_2(x)r_2(y)w_2(x)$
      * $S=r_1(y)r_1(x)r_2(x)w_1(y)r_2(y)w_2(x)$
      * $S'=r_1(y)r_1(x)w_1(y)r_2(x)r_2(y)w_2(x)$

6. Processamento de Transações

- Execução Correta de Transações -

* Três diferentes noções de equivalência de schedules
* Três diferentes noções de corretude de schedules
  * Serializabilidade por estado final (FSR)
    * Um schedule S é serializavel por estado final, se S é equivalente de estado final a algum schedule serial.
  * Serializabilidade por visão (VSR)
    * Um schedule S é serializavel por visão, se S é equivalente de visão a algum schedule serial.
  * Serializabilidade por conflito (CSR)
    * Um schedule S é serializavel por conflito, se S é equivalente de conflito a algum schedule serial.

6. Processamento de Transações

- Execução Correta de Transações -

* Qual critério a adotar???
  * Apresentam o mesmo grau de corretude
  * Diferentes graus de paralelismo
  * Permitem diferentes níveis de entrelaçamento

6. Processamento de Transações

- Execução Correta de Transações -

* Exemplo
  * $S = r_2(x)w_2(x)r_2(y)w_2(y)r_1(x)r_1(y)$
  * $S' = r_2(x)w_2(x)r_1(x)r_1(y)r_2(y)w_2(y)$
  * S' produz o mesmo estado final que S
    * S' é serializavel por estado final, então S' é correto
* Não é serializavel por visão
  * Valor de $r_1(y)$ é diferente em S e S'
* Não é serializavel por conflito
  * Porque ????

6. Processamento de Transações
- Execução Correta de Transações -

* FSR ⊃ VSR ⊃ CSR

$S' = r_2(x)w_2(x)r_1(x)r_1(y)r_2(y)w_2(y)$

6. Processamento de Transações

- Execução Correta de Transações -

* Qual critério a adotar???
  * Apresentam complexidade diferentes
  * Identificar schedules FSR ou VSR
    * Problema NP completo
      * Pior caso: n!
  * Identificar se um schedule é CSR poder ser feito em tempo polinomial

6. Processamento de Transações
- Execução Correta de Transações -

* Grafo de serialização de um schedule S
  * S está definido sobre um conjunto $\mathcal{T}=\{T_1, T_2, ..., T_n\}$ de transações
  * $SG(S)=(\mathcal{V},\mathcal{E})$
    * Grafo direcionado
    * $\mathcal{V}=\mathcal{T}$
    * $T_i \longrightarrow T_j \in \mathcal{E} \Leftrightarrow$
      * $T_i, T_j \in \mathcal{T}$,
      * $\exists\ p \in OP(T_i),\ q \in OP(T_j)$, tal que p conflita com q e $p <_S q$

6. Processamento de Transações
- Execução Correta de Transações -

* Exemplo
  * S está definido sobre $\mathcal{T}=\{T_1,T_2,T_3\}$
  * $S=r_3(y)r_1(y)r_1(x)r_2(x)w_1(y)r_2(y)r_3(x)w_2(x)w_3(z)$
  * SG(S)

6. Processamento de Transações
- Execução Correta de Transações -

* Teorema
  * $S \in CSR \Leftrightarrow SG(S)$ *não contém ciclos*
* Verificar existência de ciclos em grafos
  * Resolvido em tempo polinomial
  * $O(n^2)$, onde n representa o número de transações
* Exercício
  * Verifique a corretude do seguinte schedule:
  * $S' = r_3(y) r_1(y) r_1(x) r_2(x) w_1(y) r_2(y) w_2(x) r_3(x) w_3(z)$

6. Processamento de Transações

- Execução Correta de Transações -

* Prefixo L de um Schedule S
  * $OP(L) \subseteq OP(S)$
  * $\forall p,q \in OP(L),\ p <_L q \Leftrightarrow p <_S q$
  * $\forall q \in OP(L),\ \forall p \in OP(S),\ p <_S q \Rightarrow p \in OP(L)$
  * $S = \overbrace{r_3(y)r_1(y)\underbrace{r_1(x)r_2(x)w_1(y)}_{\mathcal{L}'}}^{\mathcal{L}}r_2(y)r_3(x)w_2(x)w_3(z)$
    * $\mathcal{L}'$ não é um prefixo de S
* Propriedade P de schedule fechada por prefixo
  * *Se P é válida para o schedule completo S, então P também é válida para qualquer prefixo de S*

6. Processamento de Transações

- Execução Correta de Transações -

* Serializabilidade por conflito é uma propriedade fechada por prefixo
  * Se algum prefixo não é serializavel por conflito
    * O schedule completo também não será!
  * Schedule incorreto
* Escalonador tem que identificar corretude de escalonamentos dinamicamente !

6. Processamento de Transações
- Confiabilidade de Escalonamentos -

* Corretude X Confiabilidade
  * $T_1 = w_1(x,2) r_1(z) c_1$; $T_2 = r_2(u) r_2(x) w_2(y, x+3) c_2$
  * $S = r_2(u) w_1(x,2) r_2(x) w_2(y, x+3) c_2$ ↯ T1
  * Após o abort de T_1
    * $r_2(x)$ torna-se uma leitura inconsistente
    * Abortar T_2
      * Violaria da semântica da operação c_2
* Execução de S é correta (serializavel), porém
  * Execução de S não é confiável

6. Processamento de Transações
- Confiabilidade de Escalonamentos -

* Recuperabilidade
  * $T_1 = w_1(x,2) r_1(z) c_1$; $T_2 = r_2(u) r_2(x) w_2(y, x+3) c_2$
  * $S_1 = r_2(u) w_1(x,2) r_2(x) r_1(z) w_2(y, x+3) c_1 c_2$
  * Regra
    * Se $w_i(x) <_S r_j(x) \Rightarrow c_i <_S c_j$
  * S_1 é recuperável
* Aborts em Cascata
  * $S_1 = r_2(u) w_1(x,2) r_2(x) w_2(y, x+3)$ T1 ↯
  * O abort de uma transação pode induzir ao abort de várias outras transações

6. Processamento de Transações
- Confiabilidade de Escalonamentos -

* Evitando Aborts em Cascata
  * $S_2 = r_2(u)w_1(x,2)r_1(z)c_1r_2(x)w_2(y, x+3)c_2$
  * Regra
    * Se $w_i(x) <_S r_j(x) \Rightarrow c_i <_S r_j(x)$
  * S_2 evita aborts em cascata
* Schedules Precisos (strict)
  * $S_3 = w_1(x,2)w_2(x,5)\ c_2$ T1
    * Não é possível recuperar o valor que x tinha antes da operação $w_1(x,2)$

6. Processamento de Transações
- Confiabilidade de Escalonamentos -

* Schedules Precisos (cont.)
  * $S_4 = w_1(x,2)c_1w_2(x,5)c_2$
  * Regra
    * Se $w_i(x) <_S p_j(x) \Rightarrow c_i <_S p_j(x)$ ou $a_i <_S p_j(x)$, onde $p_j \in \{r_j, w_j\}$
  * S_4 é preciso
* SGBDs devem garantir escalonamentos
  * Corretos ⇨ Serializaveis por conflito
  * Confiáveis ⇨ Precisos

6. Processamento de Transações
- Confiabilidade de Escalonamentos -

* Escalonamento rigoroso S
  * S é um schedule preciso e serializavel por conflito
  * Garante corretude e grau de confiabilidade em uma única regra
* Regra para escalonamento rigoroso
  * Se $p_i(x) <_S q_j(x)$, $p_i(x)$ e $q_j(x)$ estão em conflito $\Rightarrow c_i <_S q_j(x)$ ou $a_i <_S q_j(x)$

6. Processamento de Transações

- Execução Correta de Transações -

7. Protocolos para Controle de Concorrência
- Conceitos Básicos -

* Modelos de processamento de transações são implementados através de protocolos para o controle de concorrência
* Responsáveis pela produção de schedules com base em
  * Critério de corretude e um grau de confiabilidade
* Entrada: operações pertencentes a transações distintas
* Saída: Seqüência corretamente sincronizada
* Núcleo de um scheduler

7. Protocolos para Controle de Concorrência
- Conceitos Básicos -

* Classificação

- **Protocolos**
  - Conservadores
    - Bloqueio em Duas Fases (2PL)
      - Básico
      - Preciso
      - Rigoroso
    - Bloqueio Altruísta (AL)
    - Bloqueio Cooperativo (CL)
  - Agressivos
    - Pessimistas
      - Ordenamento por Marca de Tempo (TO)
      - Teste do Grafo do Serialização (SGT)
    - Otimistas
      - Certificador 2PL
      - Certificador TGS
      - Certificador TO

7. Protocolos para Controle de Concorrência
- Conceitos Básicos -

* Conservadores
  * Atrasam a execução de operações para sincroniza-las corretamente
  * Baseados em mecanismos de bloqueio
* Agressivos
  * Sincronizam operações imediatamente
  * Pessimistas
    * Decidem se aceitam ou rejeitam a operação
      * Se rejeitam, abortam a transação
  * Otimista
    * Aceitam a operação imediatamente
    * Periodicamente verificam a corretude do schedule produzido

7. Protocolos para Controle de Concorrência - Protocolos Conservadores -

* Protocolo de Bloqueio em Duas Fases - 2PL
  * Two-phase lock
  * Aquisição de bloqueio
    * Um tipo de bloqueio associado a cada objeto do banco de dados acessado por uma transação
    * Tipo de bloqueio é definido pela operação executada
      * Bloqueio de leitura (read lock - rl)
      * Bloqueio de escrita (write lock - wl)
    * SQL Server
      * syslockinfo
  * Liberação de bloqueio
    * Bloqueios de leitura e escrita

7. Protocolos para Controle de Concorrência
- Protocolos Conservadores -

* Protocolo de Bloqueio em Duas Fases - 2PL
  * Duas Fases
    * Aquisição de bloqueios
    * Liberação de bloqueios

7. Protocolos para Controle de Concorrência
- Protocolos Conservadores -

* Protocolo de Bloqueio em Duas Fases - 2PL
  * Compatibilidade de bloqueios
    * Bloqueios compatíveis
      * Podem ser concedidos simultaneamente
        * Bloqueios compartilhados
    * Bloqueios incompatíveis
      * Bloqueios exclusivos

7. Protocolos para Controle de Concorrência
- Protocolos Conservadores -

* Protocolo de Bloqueio em Duas Fases - 2PL
  * Tabela de compatibilidade de bloqueios

Transação que solicita bloqueio

Transação que retém bloqueio

	$rl_i(x)$	$wl_i(x)$
$rl_j(x)$	+	—
$wl_j(x)$	—	—

7. Protocolos para Controle de Concorrência - Protocolos Conservadores -

* Protocolo 2PL Básico

1. Escalonador recebe $p_i \in \{r, w\}$ *da transação* T_i

   / Para escalonar p_i, precisa conceder bloqueio $pl_i(x)$ */*

   Se existir bloqueio incompatível com $pl_i(x)$

   atrasa a execução de $p_i(x)$, até que $pl_i(x)$ seja concedido

   Senão

   concede $pl_i(x)$ *e escalona* $p_i(x)$

2. Uma vez uma transação T_i libere algum bloqueio, T_i não pode mais obter bloqueios

* Produz escalonamentos CSR

* Corretos
* Não evita aborts em cascata
  * Porque ?

* Protocolo 2PL Preciso

1. Escalonador recebe $p_i \in \{r, w\}$ *da transação* T_i

   / Para escalonar p_i, precisa conceder bloqueio $pl_i(x)$ */*

   Se existir bloqueio incompatível com $pl_i(x)$

   atrasa a execução de $p_i(x)$, até que $pl_i(x)$ seja concedido

   Senão

   concede $pl_i(x)$ *e escalona* $p_i(x)$

2. Ao liberar algum bloqueio de leitura, T_i não pode obter mais bloqueios. ***Bloqueios de escrita*** só serão liberados durante a execução da operação de ***commit***

* Produz escalonamentos CSR e precisos

7. Protocolos para Controle de Concorrência
- Protocolos Conservadores -

* Protocolo 2PL Rigoroso

1. Escalonador recebe $p_i \in \{r, w\}$ *da transação* T_i

   / Para escalonar p_i, precisa conceder bloqueio $pl_i(x)$ */*

   Se existir bloqueio incompatível com $pl_i(x)$

   atrasa a execução de $p_i(x)$, até que $pl_i(x)$ seja concedido

   Senão

   concede $pl_i(x)$ *e escalona* $p_i(x)$

2. T***odos os bloqueios*** só podem ser liberados durante execução da operação de ***commit***.

* Produz escalonamento Rigoroso

* Implementado pelos SGBDs existentes

7. Protocolos para Controle de Concorrência
- Protocolos Conservadores -

7. Protocolos para Controle de Concorrência
- Protocolos Conservadores -

* Execução de commit
  * Não é uma operação atômica

7. Protocolos para Controle de Concorrência
- Deadlock -

* Cenário ⇨ Escalonador 2PL Rigoroso
  * $T_1 = r_1(x) w_1(x) c_1$; $T_2 = r_2(x) w_2(x) c_2$

$r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)$

2PL Rigoroso

$S = r_1(x)\, r_2(x)$

  * T_2 espera que T_1 libere o bloqueio $rl_1(x)$
  * T_1 espera que T_2 libere o bloqueio $rl_2(x)$
  * **Deadlock**
* Deadlock
  * Detecção e Resolução
  * Prevenção

7. Protocolos para Controle de Concorrência
- Deadlock -

* Detecção de Deadlocks por Timeout
  * Se o tempo de espera de T_i por um bloqueio atingir um determinado limite, o scheduler supõem que T_i está envolvida em um deadlock
  * Resolução
    * T_i deve ser abortada
* Problema
  * Pode ser que T_i não estivesse envolvida em um deadlock
  * Aumentar time-out
    * A detecção de deadlocks pode demorar longos períodos de tempo

7. Protocolos para Controle de Concorrência
- Deadlock -

* Detecção de Deadlocks através do Grafo de Espera (Wait-For Graph)
  * O scheduler deve manter um grafo (WFG)
    * Nós representam as transações
    * Arestas são construída da seguinte forma:
      * Se T_i está esperando que T_j libere algum bloqueio, então
        * $T_i \rightarrow T_j$ deve ser inserida no grafo
      * Se a inclusão da aresta $T_i \rightarrow T_j$ introduzir ciclo em WFG, então
        * T_i e T_j estão envolvidas em um deadlock

7. Protocolos para Controle de Concorrência
- Deadlock -

* Grafo de Espera
  * $T_1 = r_1(x) w_1(x) c_1$ $T_2 = r_2(x) w_2(x) c_2$
  * $S = r1(x) r2(x) w_2(X) w_1(X)$

* Resolução
  * T_2 (mais recente) deve ser abortada (vítima)

7. Protocolos para Controle de Concorrência
- Deadlock -

* Prevenção de Deadlocks
  * 90% de deadlocks produzidos pela conversão de bloqueios
    * Bloqueios de leitura → Bloqueios de escrita
* Reduzir conversões de rl em wl
  * Pode reduzir a ocorrência de deadlocks
    * Redução da taxa de abort

7. Protocolos para Controle de Concorrência
- Deadlock -

* Prevenção de Deadlocks
  * Bloqueio tipo ***update***
    * Bloqueio para leitura com a intenção de executar operação de escrita no futuro
    * Ainda há necessidade de converter o bloqueio de update em bloqueio de escrita
    * Especificado pelo usuário

7. Protocolos para Controle de Concorrência
- Deadlock -

* Prevenção de Deadlocks
  * Tabela de Compatibilidade de bloqueios
    * Com bloqueio do tipo update

	$rl_i(x)$	$wl_i(x)$	$ul_i(x)$
$rl_j(x)$	+	—	—
$wl_j(x)$	—	—	—
$ul_j(x)$	+	—	—

7. Protocolos para Controle de Concorrência
- Deadlock -

* Prevenção de Deadlocks
  * Exemplo com bloqueio tipo update
    * $T_1 = ul_1(x) r_1(x) w_1(x) c_1 ; T_2 = ul_2(x) r_2(x) w_2(x) c_2$

$ul_2(x)$

* $S = ul_1(x) r_1(x) \; w1(x) c1$

7. Protocolos para Controle de Concorrência
- Deadlock -

* Prevenção de Deadlocks
  * Evitar que o sistema entre em deadlock
  * Marca de Tempo
    * timestamp (ts)
    * Índice temporal para transações
    * Se T_i inicia antes de $T_j \Rightarrow ts(T_i) < ts(T_j)$

7. Protocolos para Controle de Concorrência
- Deadlock -

* Prevenção de Deadlocks
  * Se T_i solicitar bloqueio sobre um objeto bloqueado por T_j

wait-die:

Se $ts(T_i) < ts(T_j)$,
T_i espera,
Senão abortar T_i

wound-wait:

Se $ts(T_i) < ts(T_j)$,
aborta T_j,
Senão T_i espera

7. Protocolos para Controle de Concorrência
- Deadlock -

* Prevenção de Deadlocks
  * Transações mais antigas são priorizadas
    * Transações mais antigas esperam (wait-die)
    * Transações mais antigas induzem o cancelamento de transações mais jovens (wound-wait)

7. Protocolos para Controle de Concorrência
- Deadlock -

* Prevenção de Deadlocks
  * Tratamento de transações mais jovens
    * Wait-die
      * Caso uma transação mais jovem não seja abortada
        * Torna-se mais antiga ao longo do tempo
          * Reduz a probabilidade de ser abortada
    * Wound-wait
      * Quando o scheduler tenta abortar a transação mais jovem (T_j)
        * T_j pode já ter executado commit
          * Evita seu cancelamento

7. Protocolos para Controle de Concorrência
- Livelock -

* Estratégia de resolução de deadlocks
  * Transação mais recente é escolhida como vítima para ser abortada
* Problema
  * Uma transação T_i é escolhida como vítima
    * Sempre que T_i é reiniciada, envolve-se em deadlock como transação mais recente
      * T_i é abortada continuamente
* Situação de livelock (starvation)
  * T_i nunca é completada (sempre é abortada)
* Solução
  * Reiniciar transação abortada (por deadlock) com mesma marca de tempo

7. Protocolos para Controle de Concorrência - Múltipla Granulosidade de Bloqueio -

* Granulosidade de Bloqueios
  * Quantidade de objetos envolvidos em um bloqueio
    * Granulosidade grossa
      * Bloqueio sobre uma tabela
    * Granulosidade fina
      * Bloqueio sobre uma tupla

7. Protocolos para Controle de Concorrência - Múltipla Granulosidade de Bloqueio -

* Importante parâmetro para performance do SGBD
  * Granulosidade mais grossa
    * Diminui o overhead no gerenciamento de bloqueios
      * Menor número de bloqueios
    * Reduz o grau de concorrência
  * Granulosidade mais fina
    * Alto overhead no gerenciamento de bloqueios
      * Maior número de bloqueios
        * Cada tupla na syslockinfo, ocupa 128 bytes (SQL Server)
    * Aumenta grau de concorrência

7. Protocolos para Controle de Concorrência - Múltipla Granulosidade de Bloqueio -

* 2PL com múltipla granulosidade
  * Grafo de granulosidade
    * Organização lógica do BD

7. Protocolos para Controle de Concorrência - Múltipla Granulosidade de Bloqueio -

* 2PL com múltipla granulosidade
  * Funcionamento
    * Bloqueio em um nó do grafo de granulosidade
    * Descendentes são implicitamente bloqueados
      * Mesmo tipo de bloqueio
    * Propagar para os ascendentes os efeitos do bloqueio
      * Bloqueio intencional
        * Intencional de leitura - irl
        * Intencional de escrita - iwl
        * Intencional de update - iul

7. Protocolos para Controle de Concorrência - Múltipla Granulosidade de Bloqueio -

* 2PL com múltipla granulosidade
  * Tabela de compatibilidade com bloqueios intencionais

	$rl_i(x)$	$wl_i(x)$	$ul_i(x)$	$irl_i(x)$	$iwl_i(x$	$iul_i(x)$
$rl_j(x)$	+	—	—	+	—	—
$wl_j(x)$	—	—	—	—	—	—
$ul_j(x)$	+	—	—	+	—	—
$irl_i(x)$	+	—	—	+	+	+
$iwl_i(x$	—	—	—	+	+	+
$iul_i(x)$	+	—	—	+	+	+

* 2PL com múltipla granulosidade

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Atividade lab
  * Execute o seguinte, para desligar a função de lock escalation

```
alter table nation
set (lock_escalation=disable)
```

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Atividade lab
  * Execute as seguintes transações

1
```
Begin transaction T1_2PL_R
declare @valor int
set transaction isolation level
repeatable read
select @valor=n_regionkey
 from nation with (rowlock)
where n_nationkey=1
set @valor = @valor+0
update nation with (rowlock)
set n_regionkey=@valor
where n_nationkey=1
waitfor delay '00:00:40'
commit
```

2
```
Begin transaction T2_2PL_R
set transaction isolation
level repeatable read
select * from nation with
(rowlock)
commit
```

3
```
select rsc_dbid, rsc_type,
req_mode, req_status,
req_transactionID
from syslockinfo
```

- Analise os bloqueios concedidos e em que tipos de objetos

7. Protocolos para Controle de Concorrência - Arquitetura Gerenciador de Transação 2PL -

7. Protocolos para Controle de Concorrência
- Protocolos Agressivos -

* Não atrasam a execução de operações
  * Não são baseados em bloqueios
* Pessimista
  * Ordenamento por marca de tempo
  * Teste do Grafo de Serialização
* Otimistas
  * Certificação 2PL
  * Certificação TGS
  * Certificação TO

7. Protocolos para Controle de Concorrência - Protocolos Agressivos [Pessimistas] -

* Ordenamento por marca de tempo (TO)
  * Cada transação T_i possui uma marca de tempo $TS(T_i)$
    * $TS(T_i)$ é associada a cada operação de T_i
  * Ao receber uma operação $q_j(x)$
    * Se $p_i(x) < q_j(x)$, $p_i(x)$ e $q_j(x)$ em conflito
      * Se $TS(T_i) < TS(T_j)$ então escalonar $q_j(x)$
      * Caso contrário, abortar T_j
  * Garante serializabilidade por conflito
    * Garante a ordem de serialização com base na ordem temporal das transações

7. Protocolos para Controle de Concorrência
- Protocolos Agressivos [Pessimistas] -

* Ordenamento por marca de tempo
  * Exemplo

7. Protocolos para Controle de Concorrência
- Protocolos Agressivos [Pessimistas] -

* Teste do Grafo de Serialização
  * Gerenciamento do grafo de serialização
    * Garantir a inexistência de ciclos
  * Protocolo

Para cada nova transação T_i, incluir um nó no grafo;

Para cada operação $p_i(x) \in OP(T_i)$ recebida, faça

Se existe $q_j(x) \in OP(T_j)$ que conflite com $p_i(x)$, já escalonada

Incluir aresta $T_j \rightarrow T_i$

Se a aresta $T_j \rightarrow T_i$ introduz um ciclo no grafo

Abortar T_i

7. Protocolos para Controle de Concorrência
- Protocolos Agressivos [Pessimistas] -

* Teste do Grafo de Serialização
  * Grafo mantido em memória
  * Processo de coleta de lixo
    * Se o nó correspondente a uma transação terminada Ti só possui arestas saindo
      * O nó pode ser removido do grafo
    * Porque??
      * Desafio: 0,3 na 2a. Aval.

7. Protocolos para Controle de Concorrência
- Protocolos Agressivos [Otimistas] -

* Certificadores 2PL
  * Quando o escalonador recebe o commit de T_i (c_i), verifica

    Se $\exists p \in OP(T_i)$ e $\exists q \in OP(T_j)$, onde p conflita com q, T_j é uma transação ativa

    Abortar T_i

    Senão

    Executar c_i

7. Protocolos para Controle de Concorrência
- Protocolos Agressivos [Otimistas] -

* Certificadores 2PL no SQL Server
  * Para cada alteração em uma tupla (operação de escrita)
    * Verifica se outra transação alterou valor lido da tupla a ser alterada
      * Compara valores de todas as colunas da tupla
      * Compara uma marca de tempo que pode se associada às tuplas de uma tabela

7. Protocolos para Controle de Concorrência
- Protocolos Agressivos [Otimistas] -

* Certificadores TGS
  * Mantém um grafo de serialização (SG), como no protocolo TGS
  * Quando o escalonador recebe o commit de T_i (c_i), verifica
    Se T_i está envolvida em um ciclo no SG
        Abortar T_i
    Senão
        Executar c_i

7. Protocolos para Controle de Concorrência
- Protocolos Agressivos [Otimistas] -

* Certificadores TO
  * Escalonador aplica a regra do protocolo TO
  * Quando o escalonador recebe o commit de T_i (c_i), verifica

    Se $\forall p,q$, $p \in OP(T_i)$, $q \in OP(T_j)$, p e q estão em conflito, $TS(T_i) < TS(T_j)$ e $p <_s q$

    Executar c_i

    Senão

    Abortar T_i

7. Protocolos para Controle de Concorrência
- Protocolos Agressivos [Otimistas] -

* Protocolos não baseados em bloqueios
  * Vantagens
    * Maior grau de concorrência, pois permitem mais entrelaçamentos
    * Não sofrem do problema de deadlocks
  * Desvantagens
    * Alta taxa de aborts
    * Não garantem grau de confiabilidade
      * Deve-se implementar a regra correspondente ao grau a ser garantido

7. Protocolos para Controle de Concorrência - Mecanismo de Múltiplas Versões [Duas] -

* Princípio de Funcionamento
  * Uma operação de escrita de T sobre um objeto x gera x_n
    * Nova versão para x
    * Só T "enxerga" x_n
  * Para executar a operação de escrita
    * T obtém primeiro um bloqueio de escrita sobre x
      * Após obter wl(x), w(x) é escalonada (e executada)
        * T gera a versão x_n
    * Qualquer outra transação pode ler a versão x
      * x é a versão realmente armazenada no BD

7. Protocolos para Controle de Concorrência - Mecanismo de Múltiplas Versões [Duas] -

* Princípio de Funcionamento
  * Quando T executa seu commit, x_n torna-se a versão x
    * Existente no banco de dados
* Redução da taxa de bloqueios
  * Leituras sobre x não são atrasadas devido a uma escrita sobre x
  * Escritas sobre x não são atrasadas devido a uma leitura sobre x

7. Protocolos para Controle de Concorrência - Mecanismo de Múltiplas Versões [Duas] -

* Protocolo 2V2PL (two version 2PL)
  * Tabela de compatibilidade
    * Novo tipo de bloqueio
      * Certify lock

	$rl_i(x)$	$wl_i(x)$	$cl_i(x)$
$rl_j(x)$	+	+	—
$wl_j(x)$	+	—	—
$cl_j(x)$	—	—	—

Certify Lock

7. Protocolos para Controle de Concorrência - Mecanismo de Múltiplas Versões [Duas] -

* Protocolo 2V2PL

1. Scheduler recebe $w_j(x)$

   Tenta obter $wl_j(x)$

   Se existe $wl_k(x)$ ou $cl_k(x)$, então

   Aguarda a concessão de $wl_j(x)$ e não escalona $w_j(x)$

   Senão

   Concede o bloqueio $wl_j(x)$

   Converte $w_j(x)$ em $w_j(x_n)$

   Escalona $w_j(x_n)$

7. Protocolos para Controle de Concorrência - Mecanismo de Múltiplas Versões [Duas] -

* Protocolo 2V2PL

2. Scheduler recebe $r_j(x)$
   Tenta obter $rl_j(x)$
   Se existe $cl_k(x)$, então
      Aguarda a concessão de $rl_j(x)$ e não escalona $r_j(x)$
   Senão
      Concede o bloqueio $rl_j(x)$
      Se T_j possuía $wl_j(x)$ // executou $w_j(x_n)$
         Converte $r_j(x)$ em $r_j(x_n)$
         Escalona $r_j(x_n)$
      Senão
         Escalona $r_j(x)$

7. Protocolos para Controle de Concorrência - Mecanismo de Múltiplas Versões [Duas] -

* Protocolo 2V2PL

3. Scheduler recebe a operação c_j

// Tenta Converter todos wl_j em cl_j

Enquanto houver $wl_j(x)$, faça

Se existir $rl_k(x)$, com $0<K\leq n$. $k \neq j$

Aguarda a concessão de $cl_j(x)$

Senão

Concede o bloqueio $cl_j(x)$

Fim-enquanto

Escalona c_j

7. Protocolos para Controle de Concorrência - Mecanismo de Múltiplas Versões [Duas] -

* Produz schedules serializaveis por conflito e precisos
* Maior grau de paralelismo
* Muitos DBMSs existentes já implementam
  * Oracle (read consistency isolation)
  * Illustra (Informix)
  * SQL Server (snapshot)

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Bloqueios em SQL

- Granulosidade
- Tipo de bloqueio
- Nivel de Isolamento
  - Duração
    - Curta
    - Longa
  - Escopo
    - Objeto
    - Predicado

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Granulosidade de bloqueios
  * DB2
    * Objetos
      * Tablespace
      * Table
      * Page
      * Row
    * Definição
      * create tablespace ... locksize <objeto | any>
        * any
          * Escalonamento dinâmic pelo DB2
            * Tablespace ↔ page

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Granulosidade de Bloqueios
  * SQL Server
    * Objetos
      * Banco de dados
      * Arquivo
      * Tabela
      * Estrutura de índice
      * Extensão (extent)
        * Conjunto de oito páginas contíguas

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Granulosidade de Bloqueios
  * SQL Server
    * Objetos
      * Página
      * Chave de busca (key)
        * Utilizado para bloquear tuplas pela estrutura de índice
          * Intervalo de chave de busca
      * Tupla (RID)

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Granulosidade de bloqueios
  * SQL Server
    * Definição
      * No código SQL da transação
    * Escalonamento automático de granulosidade de bloqueio
  * Oracle
    * Não implementa múltipla granulosidade de bloqueio

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Modo de bloqueio
  * Leitura (share - S)
    * Definido automaticamente pelo tipo de operação
  * Escrita (exclusive - X)
    * Definido automaticamente pelo tipo de operação
    * Especificado pelo usuário
  * Leitura com intenção de escrita (update - U)
    * Especificado pelo usuário

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Nível de isolamento
  * Escopo de bloqueio
    * Objeto
      * Bloqueio sobre tuplas existentes na tabela
    * Predicado
      * Bloqueio sobre tuplas que satisfazem um predicado
        * Tuplas existentes no banco de dados
        * Tuplas ainda não existentes
          * Satisfariam o predicado se incluídas no banco de dados
          * Tuplas a serem alteradas e que passariam a satisfazer o predicado após a alteração

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Nível de isolamento
  * Duração do bloqueio
    * Longa
      * O bloqueio é retido até após o commit ou abort
    * Curta
      * O bloqueio é liberado imediatamente após a execução da operação associada ao bloqueio

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Níveis de Isolamento
  * read uncommitted
    * Não existe bloqueio de leitura
    * Bloqueios de escrita
      * Curta duração tanto para objetos como predicados
  * read committed Default: Oracle e SQL Server
    * Bloqueios de leitura
      * Curta duração tanto para objetos como predicados
    * Bloqueios de escrita
      * Longa duração tanto para objetos como predicados

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Níveis de Isolamento
  * read committed
    * Anomalias
      * Nonrepeatable read
        * $S_1 = r_2(x)w_1(x,2)c_1r_2(x)c_2$
          * Transação T2 está lendo valores diferentes de x
          * Deve ser evitado !
      * Lost update
        * $S_2 = r_1(x)r_2(x)w_2(x, x+10)c_2w_1(x,x+20)r1(z)c_1$
          * Operação de escrita $w_2(x)$ foi sobreposta por $w_1(x)$

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Níveis de Isolamento
  * cursor stability
    * Bloqueios de leitura
      * Curta duração para objetos e predicados
      * Curta duração Cursor/Resultset
        * Bloqueio no item corrente do cursor é mantido até
          * Posição no Cursor/Resultset seja alterada
          * Cursor/ResultSet seja fechado
    * Bloqueios de escrita
      * Longa duração para objetos e predicados

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Níveis de Isolamento
  * cursor stability
    * Evita a anomalia lost update
      * $S_2 = r_1(x) r_2(x) \overbrace{w_2(x, x+10) c_2} w_1(x, x+20) r_1(z) c_1$
    * SGBDs que implementam
      * DB2
        * Nível de isolamento
      * SQL Server
        * Na clausula DECLARE CURSOR Opção SCROLL_LOCKS

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Níveis de Isolamento
  * repeatable read
    * Bloqueios de leitura
      * Longa duração para objetos
      * Curta duração para predicados
    * Bloqueios de escrita
      * Longa duração para objetos e predicados
    * Anomalia
      * Fenômeno Fantasma
        * T acessa tuplas que satisfazem um predicado P
          * Tuplas existentes no BD e que satisfazem P são bloqueados
          * Novas tuplas que satisfazem P podem ser inseridas

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Níveis de Isolamento
  * serializable
    * Bloqueios de leitura
      * Longa duração para objetos e predicados
    * Bloqueios de escrita
      * Longa duração tanto para objetos como predicados

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Sintaxe SQL99

* SET TRANSACTION | READ ISOLATION LEVEL | READ UNCOMMITTED
  READ WRITE | READ COMMITTED
  CURSOR STABILITY
  REPEATABLE READ
  SERIALIZABLE

DB2

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Sintaxe Oracle

* SET TRANSACTION
  - ISOLATION LEVEL
    - READ COMMITTED
    - SERIALIZABLE
  - READ
  - READ WRITE

READ
  Transação não pode alterar o banco de dados
  Assume Transaction-level Read Consistency (default)
READ WRITE
  Transação pode alterar o banco de dados
  Assume Transaction-level Read Consistency (default)

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Sintaxe SQL Server (>2005)
  * SET TRANSACTION ISOLATION LEVEL
    - READ UNCOMMITTED
    - READ COMMITTED
    - REPEATABLE READ
    - SNAPSHOT
    - SERIALIZABLE
  * Snapshot
    * Implementa o protocolo 2V2PL
    * Necessário ligar opção ALLOW_SNAPSHOT_ISOLATION

7. Protocolos para Controle de Concorrência - Controle de Concorrência em SQL -

* Sintaxe SQL Server (versão > 2005)
  * Protocolo otimista (Cursores - declare cursor)
    * OPTIMISTIC WITH VALUES
      * Ao executar uma operação de escrita
        * Escalonado verifica se valores atuais de todas colunas da tupla são os mesmos de quando a tupla foi lida
          * Se não for verdade, a transação é abortada
    * OPTIMISTIC WITH ROW VERSIONING
      * Compara valores da coluna timestamp
        * Criar tabela com esta coluna
  * Ligar opção READ_COMMITTED_SNAPSHOT
    * Nível de isolamento read committed

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Escolha do nível de isolamento
  * Importante decisão de projeto
  * Decidir entre
    * Garantir consistência do banco de dados
    * Garantir um alto grau de concorrência

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Atividade lab
  * Execute o seguinte comando DDL

```
alter table nation
set (lock_escalation=disable)
```

  * Execute a transação

```
Begin transaction T1
set transaction isolation level repeatable read
select * from nation with (rowlock)
waitfor delay '00:00:20'
commit
```

  * Dispare a consulta

```
select rsc_dbid, rsc_type, req_mode, req_transactionID
from syslockinfo
```

  * Analise os bloqueios concedidos e em que tipos de objetos

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Atividade lab
  * Execute a transação

```
Begin transaction T1
set transaction isolation level repeatable read
select * from nation with (updlock, rowlock)
waitfor delay '00:00:40'
commit
```

  * Dispare a consulta

```
select rsc_dbid, rsc_type, req_mode,
req_transactionID
from syslockinfo
```

  * Analise os bloqueios concedidos e em que tipos de objetos

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Atividade lab
  * Execute o seguinte comando DDL

```
alter table lineitem
set (lock_escalation=disable)
```

  * Execute a transação

```
Begin transaction T1
set transaction isolation level repeatable read
select * from lineitem with (rowlock)
waitfor delay '00:00:10'
commit
```

  * Dispare a consulta

```
select rsc_dbid, rsc_type, req_mode, req_transactionID
from syslockinfo
```

  * Analise os bloqueios concedidos e em que tipos de objetos

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Atividade lab
  * Execute a transação

```
Begin transaction T1
set transaction isolation level repeatable read
select * from lineitem
waitfor delay '00:00:10'
commit
```

  * Dispare a consulta

```
select rsc_dbid, rsc_type, req_mode,
req_transactionID
from syslockinfo
```

  * Analise os bloqueios concedidos e em que tipos de objetos

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Atividade lab
  * Execute a transação

```
begin transaction T1
set transaction isolation level repeatable read
select * from lineitem with (rowlock)
waitfor delay '00:00:30'
update lineitem with (rowlock)
set l_quantity=l_quantity
commit
```

  * Dispare a transação

```
begin transaction T2
set transaction isolation level repeatable read
select * from lineitem with (tablock)
update lineitem with (tablock)
set l_quantity=l_quantity
commit
```

  * Analise o que ocorre durante a execução das duas transações

* Atividade lab A

* Execute a transação

```
begin transaction T1
set transaction isolation level repeatable read
update lineitem with (paglock)
set l_quantity=l_quantity
waitfor delay '00:00:30'
commit
```

* Dispare a transação

```
begin transaction T2
set transaction isolation level repeatable read
select * from lineitem with (paglock)
commit
```

* Atividade lab B

* Execute a transação

```
alter database tpc_h set allow_snapshot_isolation on
begin transaction T1
set transaction isolation level snapshot
update lineitem with (paglock)
set l_quantity=l_quantity
waitfor delay '00:00:30'
commit
```

* Dispare a transação

```
alter database tpc_h set allow_snapshot_isolation on
begin transaction T2
set transaction isolation level snapshot
select * from lineitem with (paglock)
commit
```

7. Protocolos para Controle de Concorrência
- Controle de Concorrência em SQL -

* Desafio (0,5 na 2a. aval)
  * Justificar o que ocorre nas atividades A e B

8. Recuperação e Logging
- Conceitos Básicos -

* Recuperação
  * Manipulação automática pelo SGBD de todos os tipos de falhas esperados
* Condição básica
  * Armazenamento de informação redundante durante o funcionamento normal (logging)
* Paradigma de Transação
  * Unidade de recuperação (propriedade tudo ou nada)
  * Persistência após uma execução com sucesso
* Estado do BD que deve ser recuperado
  * Estado consistente mais recente antes da falha

8. Recuperação e Logging
- Conceitos Básicos -

* UNDO
  * Desfazer por completo os efeitos das operações das transações que não haviam executado commit no momento da falha
    * Alterações já tenham sido transferidos para disco
    * Até o início da transação
* Savepoint
  * Desfazer o efeito das operações até um determinado ponto dentro da transação
    * Oracle, SQL Server, DB2, Postgres, MySQL

8. Recuperação e Logging
- Conceitos Básicos -

* REDO
  * Refazer por completo os efeitos de transações que já haviam executado commit no momento da falha
  * Alterações ainda não haviam sido descarregadas para disco
  * Banco de Dados

8. Recuperação e Logging
- Tipos de Falhas -

* Falha de uma Transação ativa (abort)
  * UNDO
* Falha de sistema
  * Recuperar o mais recente estado consistente do BD que existia antes da ocorrência da falha
  * UNDO
  * REDO

8. Recuperação e Logging
- Tipos de Falhas -

* Falha de Media (recuperação de disco)
  * Refazer o BD utilizando o último backup
  * RAID
* Catástrofe
  * Uma ambiente computacional espelho

8. Recuperação e Logging
- Arquitetura do Gerenciador de Recuperação -

* Disk Log File
  * Transaction failure
  * System failure
    * DB + Disk Log File ⇒ DB
* Media failure (long duration)
    * Backup + Tape Log File + Disk Log File ⇒ DB

8. Recuperação e Logging
- Técnicas de Logging -

* Logging Físico
  * Before images e after images de páginas alteradas são armazenados no arquivo de log
  * Granulosidade do log
    * Página
      * Se apenas uma tupla da página p é alterada
        * Before image e after image de p são armazenados no log
* UNDO
  * Assumir valores do before image
* REDO
  * Assumir valores do after image

8. Recuperação e Logging
- Técnicas de Logging -

* Logging Físico
  * Processo de recovery rápido e fácil de ser implementado
  * Exige muito espaço para o arquivo de log
  * Utilizado pela maioria dos DBMSs existentes

8. Recuperação e Logging
- Técnicas de Logging -

* Logging Lógico
  * Comandos DML (update, insert, delete) com seus parâmetros são protocolados no arquivo de log
  * UNDO
    * Executar operação inversa da armazenada no arquivo de log
      * Executar delete para cada insert no log
      * Um update equivale
        * Delete tupla com valor antigo + Insert tupla com novo valor
  * REDO
    * Executar todos os comandos DML protocolados no log

8. Recuperação e Logging
- Técnicas de Logging -

* Logging Lógico
  * Utiliza menos espaço no arquivo de log
  * Processo de recovery complexo e longo

8. Recuperação e Logging
- Técnicas de Logging -

* Logging Fisico-lógico
  * Granulosidade do log (Físico)
    * Página
  * Para cada página (Lógico)
    * Armazenar alterações em tuplas de uma página como comandos DML
  * UNDO
    * Executar operação inversa da armazenada no log
  * REDO
    * Executar todos os comandos DML protocolados no log

8. Recuperação e Logging
- Técnicas de Logging -

* Logging Fisico-Lógico
  * Executar operações de UNDO e REDO por página
    * Menor número de execuções que logging lógico
    * Checkpoint e teste para verificar necessidade de REDO

8. Recuperação e Logging
- Técnicas de Logging -

* Estratégias para identificar o momento de gravar registros de log
  * Estratégia WAL (Write-Ahead-Log)
    * Antes de transferir uma página P da area de buffer para o BD, gravar no arquivo de log registros referentes às alterações sobre P
  * Estratégia Force-Log-at-Commit
    * Antes de terminar a execução da operação de commit de uma transação T, gravar no arquivo de log registros referentes às alterações executadas por T em páginas ainda residente em buffer

8. Recuperação e Logging
- Técnicas de Logging -

* Estratégia de troca de página no buffer
  * Steal
    * Páginas alteradas podem ser trocadas e descarregadas no BD a qualquer momento
      * mesmo antes do EOT
    * Maior flexibilidade para a política de substituição de páginas
    * Necessário UNDO
  * NoSteal:
    * Páginas alteradas são mantidas no buffer até o EOT
    * Não é necessário UNDO
    * Problemas com transações longas

8. Recuperação e Logging
- Técnicas de Logging -

* Estratégia de processamento de EOT (commit)
  * Force
    * Páginas alteradas são descarregadas no BD durante o processamento de EOT (transação que fez a alteração)
    * Não é necessário REDO
    * Muitas escritas em disco são necessárias
  * NoForce:
    * Durante o processamento de EOT, nenhuma propagação de página é disparada
    * Pouco overhead com escritas em disco durante EOT
    * É necessário REDO

8. Recuperação e Logging
- Técnicas de Logging -

* Bloqueio
  * Granularidade de log ≤ Granularidade de bloqueio
  * Porque?
    * Slide 384

8. Recuperação e Logging

* Atividade lab (rollback/savepoint)
  * Execute a seguinte transação

```
begin transaction T1
declare @valor int
select @valor=qtde from Produtos
 where id_produto=1
set @valor=@valor+20
update Produtos set qtde=@valor where id_produto=1
save transaction SV1  -- Definição savepoint SV1
select @valor=qtde from Produtos
 where id_produto=1
set @valor=@valor-10
update Produtos set qtde=@valor where id_produto=1
save transaction SV2 -- Definição savepoint SV2
select @valor=qtde from Produtos
 where id_produto=1
print @valor
rollback transaction SV1
commit
```

  * Descreva as ações que foram executadas pelo SGBD
    * Verifique o estado do banco de dados gerado

8. Recuperação e Logging
- Checkpoint -

* Mecanismo para limitar a quantidade de REDOs após falha do sistema

8. Recuperação e Logging
- Checkpoint -

* Recuperação
  * REDO precisa retornar muito longe no arquivo de log
* Checkpoint
  * Armazenamento de informação no arquivo de log
  * Reduzir e limitar a quantidade de REDOs após uma falha
* Onde armazenar??
  * Diferentes estratégias
    * Gravar no arquivo de log
    * Gravar no arquivo de log e no BD

8. Recuperação e Logging
- Checkpoint -

* Transaction Consistent Checkpoint (TCC)

* REDO para T_3

* Fuzzy Checkpoint

- Gravar registro de log do tipo ***Checkpoint***
  - Menor LSN de operação de atualização sobre páginas em buffer (MinDirtyPageLSN).
  - Desconsidera-se LSNs de operações sobre páginas já descarregadas

8. Recuperação e Logging
- Checkpoint -

* Fuzzy Checkpoint (cont.)
  * Identificação da posição de início das operações de REDO
    * Encontrar no log o registro do último checkpoint
    * Identificar o MinDirtyPageLSN
    * Iniciar REDO a partir deste ponto
* Descarga de alterações é realizada de forma assíncrona com relação a geração de checkpoint

8. Recuperação e Logging
- Checkpoint -

* Estratégia de Checkpoint SQL Server
  * Descarga para disco de todas páginas alteradas e residentes na área de buffer
    * Independentemente da alteração ter sido feita por transação "committed" ou não
  * Aplica a regra WAL

8. Recuperação e Logging
- Arquivo de log -

* Arquivo seqüencial
  * Cada registro de log é inserido no final do arquivo
  * Captura ordem temporal de execução das operações de transações concorrentes
* Independente do Banco de Dados
  * Replicado

8. Recuperação e Logging
- Arquivo de log -

* Tipos de registros de log
  * Begin-of-Transaction (BOT)
  * Operação de escrita
    * Informação de UNDO (before image) e de REDO (after image)
  * Commit ou abort
  * Checkpoint
    * Begin-Checkpoint
    * Dados do Checkpoint
    * End-Checkpoint

8. Recuperação e Logging
- Arquivo de Log -

* Estrutura de um registro de log
  * Log Sequence Number (LSN)
    * Identifica univocamente um registro de log
    * Cresce de forma monotônica
      * Capturar a ordem cronologica do registro de log
      * Refletir a ordem em que as alterações ocorreram
  * TRID
    * Transaction ID
  * PageID
    * Endereço da página alterado em disco

8. Recuperação e Logging
- Arquivo de log -

* Estrutura de um registro de log
  * Se a estratégia é logging físico
    * Before image
    * After image
  * Se a estratégia é logging físico-lógico ou lógico
    * Comandos DML (insert, delete, update)
  * PrevLSN da mesma transação
    * Lista encadeada para trás
* Para cada transação
  * Vários registros de log
    * BOT, um para alteração sobre o BD, savepoint, commit/rollback

8. Recuperação e Logging
- Arquivo de log -

* Atividade de lab
  * No BD Estoque, execute a seguinte consulta

```
select [Current LSN], [Operation],
[Transaction Name],
[Page ID], [Lock Information]
from fn_dblog(null,Null)
```

8. Recuperação e Logging
- Arquivo de log -

* Atividade de lab
  * Execute a seguinte transação

```
begin transaction Teste_Arq_Log
declare @valor int
select @valor=qtde from Produtos
 where id_produto=1
set @valor=@valor+20
update Produtos set qtde=@valor where id_produto=1
save transaction SV1  -- Definição savepoint SV1
select @valor=qtde from Produtos
 where id_produto=1
set @valor=@valor-10
update Produtos set qtde=@valor where id_produto=1
save transaction SV2 -- Definição savepoint SV2
select @valor=qtde from Produtos
 where id_produto=1
commit
```

  * Execute novamente a consulta para acessar o arquivo de log

8. Recuperação e Logging
- Arquivo de log -

* Atividade de lab
  * Execute a consulta para ler o arquivo de log
  * Execute a seguinte transação

```
begin transaction Teste_Arq_Log
declare @valor int
select @valor=qtde from Produtos
 where id_produto=1
set @valor=@valor+20
update Produtos set qtde=@valor where id_produto=1
save transaction SV1  -- Definição savepoint SV1
select @valor=qtde from Produtos
 where id_produto=1
set @valor=@valor-10
update Produtos set qtde=@valor where id_produto=1
save transaction SV2 -- Definição savepoint SV2
if @valor=@valor
   rollback
else
   commit
```

  * Execute novamente a consulta para acessar o arquivo de log

8. Recuperação e Logging
- Procedimentos de Recuperação -

* Quando executar REDO e UNDO ?
  * Registros de Log de transações que executaram commit antes da falha
    * Transações vencedoras
    * Podem precisar de REDO
    * Quais registros ?
      * Protocolam operações de alterações em páginas
  * Registros de Log de transações que não executaram commit
    * Transações perdedoras
    * Podem precisar de UNDO
    * Quais registros ?
      * Protocolam operações de alterações em páginas

8. Recuperação e Logging
- Procedimentos de Recuperação -

* Premissa para identificar quando executar REDO/UNDO
  * Quando uma página P é propagada para disco
    * LSN do último registro de log referente à atualização de p
      * Gravado no cabeçalho da página p
      * LSN(P)

8. Recuperação e Logging
- Procedimentos de Recuperação -

* Quando executar um REDO ?
  * LSN da página < LSN do registro de log para REDO
  * PageLSN(p) ← LSN do registro
* Quando executar um UNDO ?
  * LSN da página ≥ LSN do registro de log para UNDO
  * PageLSN(p) ← LSN do registro

8. Recuperação e Logging
- Procedimentos de Recuperação -

* Exemplo
  * Considere duas transações T_1 e T_2 que atualizam objetos de P

- Ações de Recuperação
  - UNDO da alteração em P protocolada em LSN(90)
    - PageLSN(P)=90
  - Não é necessário REDO
    - PageLSN(P)=90

8. Recuperação e Logging
- Procedimentos de Recuperação -

* Procedimento genérico (Update-in-place, Steal, NoForce,TCC)
  1. Varredura de Análise
     * Ler arquivo de log a partir do último checkpoint até fim do arquivo
     * Identificar as transações vencedoras (committed) e as perdedoras
       * Páginas alteradas por estas transações

8. Recuperação e Logging
- Procedimentos de Recuperação -

* Procedimento genérico
  2. Varredura de UNDO
     * Ler arquivo de log de trás para frente, a partir do EOF até BOT da mais antiga transação perdedora
     * Para cada transação perdedora
       * Se LSN da página ≥ LSN do registro
         * Desfazer
           * Gravar campo Before Image do arquivo de log sobre a página em disco
         * Atualizar LSN da página
           * PageLSN(p) ← LSN do registro

* Procedimento genérico
  3. Varredura de REDO
     * A partir do último checkpoint até fim do arquivo de log
     * Para cada transação vencedora
       * Se LSN da página < LSN do registro
       * Refazer
         * Gravar campo After Image do arquivo de log sobre a página em disco
       * Atualizar LSN da página
         * PageLSN(p) ← LSN do registro

8. Recuperação e Logging
- Procedimentos de Recuperação -

* Procedimento genérico (Update-in-place, Steal, NoForce,TCC)

Procedimento Idempotente:

Mesmo em situações de falha durante sua execução, produz o mesmo resultado sobre o Banco de Dados

8. Recuperação e Logging
- Procedimentos de Recuperação -

* Fase de Undo deve ocorrer antes de Redo
  * Exemplo de cenário Redo antes do Undo
    * T_1 e T_2 que atualizam objetos de P

Procedimentos de recuperação:

REDO

After Image de P; PageLSN(P)=90

UNDO

Before image de P; PageLSN(P)=60

Desfaz a alteração de T_2 ⇒ Estado inconsistente

8. Recuperação e Logging
- Procedimentos de Recuperação -

* Granulosidade log ≤ Gran. bloqueio
  * log=página; bloqueio=tupla

Procedimentos de recuperação:

UNDO: Before Image de P (r,s); PageLSN(P)=60

REDO: After image de P (r',s'); PageLSN(P)=90

s' deve ser desfeito⇒Estado inconsistente

8. Recuperação e Logging
- Procedimentos de Recuperação -

* Atividade lab
  * Consulte o log de Produtos
  * Dispare as seguinte transação sobre o BD Estoque

```
begin transaction T1
declare @valor int
select @valor=qtde from Produtos
  where id_produto=1
set @valor=@valor+20
update Produtos set qtde=@valor where id_produto=1
commit
begin transaction T2
declare @valor int
select @valor=qtde from Produtos
  where id_produto=1
set @valor=@valor-10
update Produtos set qtde=@valor where id_produto=1
select @valor=qtde from Produtos
  where id_produto=1
print @valor
waitfor delay '00:01:00'
commit
```

  * Encerre o SQL Server e, após isto, reinicie-o novamente
  * Descreva as ações que foram executadas pelo SGBD
    * Verifique o estado do banco de dados gerado
  * Consulte o log de Estoque

9. Performance de SBDs
- Benchmark -

* Diferentes DBMSs
  * Diferentes implementações
    * Diferentes performance para diferentes cargas de trabalho
* *Benchmark*
  * Quantificar performance de DBMSs
    * Número de transações executadas por segundo para uma determinada carga de trabalho
      * Vazão (*throughput*)

9. Performance de SBDs
- Benchmark -

* Classe de Aplicações de Bancos de Dados
  * Online transaction processing (OLTP)
    * Alto grau de concorrência
    * Técnicas eficientes para otimizar o processamento de commits
  * Suporte a decisão e Online Analytical Processing (OLAP)
    * Algoritmos eficientes para a execução de consultas
    * Otimização de consultas

9. Performance de SBDs
- Benchmark -

* Benchmarks TPC (Transaction Processing Performance Council - http://www.tpc.org)
  * TPC-A
    * Quantificar performance em BDs com atualizações intensivas (Aplicações OLTP)
    * Incorpora comunicação com terminais
    * Simula aplicação bancária
  * TPC-B
    * Quantificar performance do núcleo de BDs com atualizações intensivas (Aplicações OLTP)
    * TPC-A removendo a comunicação com terminais

9. Performance de SBDs
- Benchmark -

* Benchmarks TPC (cont.)
  * TPC-C
    * Modela um sistema de vendas de um fornecedor, com produtos distribuídos em vários depósitos
      * Entrada de solicitações (entrada de solicitações e entrega de encomendas, armazenamento de faturas e pagamentos)
    * Banco de dados
      * 9 tabelas
    * Carga de trabalho
      * 5 transações
        * novo pedido, pagamento, status pedido, entrega, status do estoque

9. Performance de SBDs
- Benchmark -

* Benchmarks TPC
  * TPC-E
    * Simula um ambiente de transações em bolsas de valores
    * Banco de dados
      * 33 tabelas
      * Populated with pseudo-real data
        * Distributions based on 2000 U.S. and Canada census data
          * Used for generating name, address, gender, etc.
          * Introduces natural data skew
        * Actual listings on the NYSE and NASDAQ
    * Carga de trabalho
      * 6 RO transactions
      * 4 RW Transactions

9. Performance de SBDs
- Benchmark -

* Benchmarks TPC
  * TPC-H
    * Quantificar performance de BDs com consultas intensivas (Aplicações de suporte a decisão)
      * Data warehouse application (snow flake)
    * Banco de dados
      * 8 tables
      * Several scale factors
        * SF 10: ~13GB database
    * Carga de trabalho
      * 22 queries

9. Performance de SBDs
- Sintonia Fina -

* Ajustar performance de um SGBD
  * Ajustar diversos parâmetros e decisões de projeto
    * Melhorar a performance do sistema para uma determinada aplicação
* Gargalo
  * Recurso do sistema que está sendo utilizado em sua capacidade máxima, limitando a vazão do sistema
* Sintonia fina (tuning)
  * Identificação e eliminação de gargalos

9. Performance de SBDs
- Sintonia Fina -

* Três níveis
  * Hardware
  * Parâmetros do DBMS
  * Projeto

9. Performance de SBDs
- Sintonia Fina -

* Tuning a nível de Hardware
  * Adicionar mais disco
    * Quando o gargalo é o I/O de disco
    * RAID
  * Adicionar mais memória principal
    * Quando o gargalo é área de buffer
  * Utilizar CPUs mais rápidas ou dedicadas
    * CPU só para processar consultas (DB2)

9. Performance de SBDs
- Sintonia Fina -

* Tuning a nível de parâmetros do DBMS
  * Aumentar o tamanho da área de buffer (SGBD)
    * Reduzir acesso a disco
    * Com base na taxa de acerto
    * Parâmetro de instalação do SGBD
      * Pode ser feito automaticamente por certos DBMS
  * Componente de Recovery
    * Disco de log não deve conter nenhum outro tipo de dados
    * Intervalo de geração de checkpoints
      * Aumentar ⇒ Reduz gargalo, mas aumenta tempo de Redo

9. Performance de SBDs
- Sintonia Fina -

* Tuning a nível de projeto
  * Particionamento de tabelas
    * Horizontal
      * Subconjuntos de tuplas freqüentemente acessados
      * Exemplo
        * Tabela de empregados pode ser particionada horizontalmente por lotação (departamento)

9. Performance de SBDs
- Sintonia Fina -

* Tuning a nível de projeto
  * Particionamento de tabelas
    * Vertical
      * Subconjunto de atributos freqüentemente acessados
      * Exemplo
        * Tabela de clientes de uma companhia telefônica pode ser particionada verticalmente
          * Atributos código, nome, telefone e
          * Atributos código, leitura-ant, leitura-atual

9. Performance de SBDs
- Sintonia Fina -

* Particionamento de tabela
  * Oracle
    * Suporte para particionamento horizontal
    * Exemplo
      * Particionamento de Empregado por lotação
      * Esquemas
        * Departamento(cod_dep, nome, ender)
        * Empregado(matr, nome, ender, salário, lotação)

9. Performance de SBDs
- Sintonia Fina -

* Particionamento de tabela
  * Exemplo Oracle

```
CREATE TABLE empregado
 (matr       NUMBER(9) PRIMARY KEY,
  nome      VARCHAR(50) NOT NULL,
  ender     VARCHAR(30),
  salário   DECIMAL(9,2) NOT NULL,
  lotação NUMBER(5)
  FOREIGN KEY lotação REFERENCES departamento)
  PARTITION BY RANGE (LOTAÇÃO)
  (PARTITION p1 VALUES LESS THAN (2) TABLESPACE Arq1
   PARTITION p2 VALUES LESS THAN (3) TABLESPACE Arq2
   PARTITION p3 VALUES LESS THAN (4) TABLESPACE Arq3
   PARTITION p4 VALUES LESS THAN (MAXVALUE) TABLESPACE
 Arq4);
```

9. Performance de SBDs
- Sintonia Fina -

* Particionamento de tabela
  * Oracle
    * Inserção de novas tuplas em Empregado
      * Alocação automática na partição correta
    * Consulta sobre Empregado
      * Listar os salários dos empregados do departamento 2

SELECT salario FROM empregado WHERE lotação=2

SELECT salario FROM empregado PARTITION (p2)

CREATE VIEW emp-dep02 AS
*SELECT * FROM empregado PARTITION (p2)*
SELECT salario FROM emp-dep02

9. Performance de SBDs
- Sintonia Fina -

* Particionamento de tabela
  * SQL Server
    * Suporte para particionamento horizontal
    * Passos
      * Criação dos filegroups e respectivos arquivos
      * Criação da função de particionamento
      * Criação do esquema de particionamento
      * Criação da tabela particionada

9. Performance de SBDs
- Sintonia Fina -

* Particionamento de tabela
  * SQL Server
    * Criação da função de particionamento
      * Create partition function <nome_função> (<tipo_dado_atributo>) as range left | right for values (V_1,V_2,V_3, ..., V_n)
        * Devem ser definidas n+1 partições
          * Opção left
            * Primeira partição
              * Tuplas com atr_part $\leq V_1$
            * Segunda partição
              * Tuplas com V_1 < atr_part $\leq V_2$
            * Última partição (n+1)
              * Tuplas com atr_part > V_n

9. Performance de SBDs
- Sintonia Fina -

* Particionamento de tabela
  * SQL Server
    * Criação do esquema de particionamento
      * Define partições e as associa aos filegroups
      * Create partition scheme <nome_esquema> as partition nome_Função_part to (nome_fg, ...)
    * Criação da tabela particionada
      * Create table <nome_tab> (<esquema_tab>) on nome_função_part (atr_part)

9. Performance de SBDs
- Sintonia Fina -

* Atividade de lab
  * Particionar a tabela lineitem_P (TPC-H) em três partições
    * Cada partição deve estar alocada a um arquivo específico
    * Alocar tuplas nas partições com
      * Distribuição uniforme por número de tuplas por partição
    * Povoar lineitem_P com dados de lineitem

9. Performance de SBDs
- Sintonia Fina -

* Tuning a nível de projeto
  * Reconstrução de tabelas particionadas
    * Particionamento horizontal
      * $R = \cup R_i$
    * Particionamento vertical
      * $R = \bowtie R_i$

9. Performance de SBDs
- Sintonia Fina -

* Tuning a nível de projeto
  * De-normalização
    * Reduz joins
    * Simplifica o acesso
    * Quando??
      * Dados são usados frequentemente de forma conjunta (dados do empregado e nome do departamento onde está lotado)
      * Dado redundante é estável
      * Dado redundante em pouco volume

9. Performance de SBDs
- Sintonia Fina -

* Tuning a nível de projeto
  * Índices
    * Reduz table scan
    * Evita operação de sort
      * Índice ordenado
    * B^+ é mais eficiente que hash para consultas em extensão
    * Índice primário
      * Esparso
        * Um índice por página de dados
    * Candidatos para índices
      * Colunas utilizadas em cláusulas
        * where, order by, group by e joins
      * Colunas com valores únicos

9. Performance de SBDs
- Sintonia Fina -

* Tuning a nível de projeto
  * Utilização de Índices
    * Select ... from Funcionário where matr=123 or matr>540
      * Indice não é utilizado
      * Alternativa
        * Select ... from Funcionário where matr=123 union all
        * Select ... from Funcionário where matr > 540
    * Cálculo sobre coluna índice implica na não utilização do índice
      * ... Where salário /12 > 2000

* Índices Bitmap no Oracle
  * Bitmap
  * String de bits
    * Representa uma lista de TID
    * TID
      * Um número inteiro que identifica uma tupla em uma tabela com cardinalidade igual N
      * TID $\in \{0,1,2,3,..,N-1\}$

9. Performance de SBDs
- Sintonia Fina -

* Índices Bitmap no Oracle
  * Construção do bitmap
    * Se uma tupla com TID igual a k está no bitmap
      * Bit com posição igual a k deve ter valor 1
    * Se uma tupla com TID igual a k não está no bitmap
      * Bit com posição igual a k deve ter valor 0
  * Exemplo
    * Bitmap
      * 10011000010...
        * Tuplas com TID $\in \{0,3,4,9\}$ estão no bitmap

* Índices Bitmap no Oracle

* Estrutura
  * Árvore B+
    * Para cada chave de busca nas folhas
      * Um bitmap associado
  * Funções
    * $\mathcal{F}$: TID $\mapsto$ Ponteiro
    * $\mathcal{F}^{-1}$: Ponteiro $\mapsto$ TID

9. Performance de SBDs
- Sintonia Fina -

* Tuning a nível de projeto
  * Controle de Concorrência (Bloqueios)
    * Seleção da granulosidade
    * Tipo de bloqueio (read, write, update)
    * Nível de isolamento
      * Cursor stability preferível
      * Repeatable read
  * Consultas
    * Tente se antecipar ao otimizador
    * Tempo de parsing é muito alto
      * Armazenar consultas compiladas
    * Interação client-server é muito cara
      * Recupere conjuntos

* Tuning a nível de projeto
  * Consultas
    * Evite subselect, utilize sempre que possível joins
    * Selecione apenas as colunas realmente relevantes
      * Colunas adicionais ⇒ CPU adicional
    * Evite SELECT *
    * Consider a possibilidade de utilizar sort externo no lugar de order by
      * Para um grande volume de tuplas

Exemplo de Cursor

```
#include <stdio.h>
exec sql include sqlca;
int main()
{   EXEC SQL BEGIN DECLARE SECTION;
    char usuario[20], servidor_bd[20]; int cod; float pr_venda_c;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL CONNECT TO :servidor.bd USER :usuário;
    EXEC SQL DECLARE C1 CURSOR SCROLL_LOCKS
            FOR SELECT cod_item,pr_venda  FROM Estoque WHERE pr_venda < 500  FOR
     UPDATE pr_venda;
    EXEC SQL OPEN C1;
    EXEC SQL BEGIN TRANSACTION;
    EXEC SQL SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    EXEC SQL FETCH NEXT C1 INTO :cod,:pr_venda_c;
     while(SQLCODE==0)
       {pr_venda_c=pr_venda_c*1.15;
        EXEC SQL UPDATE estoque set pr_venda=:pr_venda_c where cod_item=:cod;
       EXEC SQL FETCH NEXT C1 INTO :cod,:pr_venda_c; }
     EXEC SQL CLOSE C1;
    EXEC SQL COMMIT TRANSACTION;
    EXEC SQL DISCONNECT CURRENT;  return (0); }
```

Exemplo de Cursor (2)

```
declare @cd_mun int, @cd_zona int, @cd_secao int, @cd_cargo int,
@cd_nr_candidato int, @votos int
DECLARE c_teste CURSOR scroll_locks for
        select * from ce_voto_secao with (XLOCK)
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
begin transaction
open c_teste
fetch next from c_teste
into @cd_mun, @cd_zona, @cd_secao, @cd_cargo, @cd_nr_candidato,
@votos
while @@FETCH_STATUS = 0
begin
print @cd_mun
fetch next from c_teste
into @cd_mun, @cd_zona, @cd_secao, @cd_cargo, @cd_nr_candidato,
@votos
end
close c_teste
DEALLOCATE c_teste
commit
```

Processor Architecture

- An Abstract Model -

HD Drive
- Information Encoding -

* Head
  * One head for each surface, riding close to the surface
    * Never touches the surface
  * Reads and alters magnetism
    * R/W operation

Program Operation

- Physical Write -

S1: Block *B* with ***clean*** pages (all bits set to 1)

S2: New data should be inserted into pages P_A and P_F

★ ***Program operation***

Logical Write

S3: New data should be inserted into page P_E of *B*

- ★ *Program operation can not be execute any more*
- ★ *Pages of B are rewritten in a new clean block B'*
- ★ *Page P_E is written on block B'*
- ★ *Pages P_A and P_F (block B) become* ***stale pages***

Fusion-io
© 2009

4. Indexing
- B+-Tree Anatomy -

4. Indexing

- Operations on a B+-Tree -

4. Indexing
- Operations on a B+-Tree -

SBDs Paralelos

Chip Multi-Processor

SBDs Móveis

Logical Space
Host A
Q1
Q2
Host B
Physical Space
Context-Awareness Space

SBDs Multi-Plataforma

- Cada plataforma visitada
  - Replicação do banco de dados sob demanda

Sistemas de Fluxos de Dados

SNQL

```
SELECT  r.rID, AVG(t.collectedValue), COUNT(t.collectedValue),
               AVG(h.collectedValue), COUNT(h.collectedValue)
FROM           Humidity h, Temperature t, SensorRegion r
WHERE          r.regionId = t.regionId and r.regionId = t.regionId
AND            r.initialLatitude   > 034308 AND  r.FinalLatitude   < 034316
AND            r.initialLongitude > 383151 AND  r.FinalLongitude < 383114
AND            t.collectedValue  > 40
AND            h.collectedValue < 0.7
Group by r.rID
TIME WINDOW      3600
SEND INTERVAL    120
SENSE INTERVAL   10
SCHEDULE          1   'Sept-26-2019 14:00:00'
```

R

Sistemas de Fluxos de Dados

In-network Query Processing

Table Name: dbo.lineitem

Statistics Name: _WA_Sys_00000001_7B905C75

```
Statistics for INDEX '_WA_Sys_00000001_7B905C75'.
-------------------------------------------------------------------------------

Name                                Updated                             Rows
-------------------------------------------------------------------------------
_WA_Sys_00000001_7B905C75           Apr  3 2019 12:09PM                 6001215

All Density                         Average Length                      Columns
-------------------------------------------------------------------------------
2.667591E-05                        4                                   l_orderkey

Histogram Steps
RANGE_HI_KEY                        RANGE_ROWS                          EQ_ROWS
-------------------------------------------------------------------------------
68                                  0                                   127.7049
64036                               64359.68                            314.8813
70947                               17840.05                            314.8813
90726                               25789.04                            314.8813
118595                              28137.6                             314.8813
179905                              34596.15                            314.8813
206438                              18562.69                            314.8813
221543                              19420.82                            314.8813
242886                              22175.86                            314.8813
272738                              20143.45                            314.8813
326625                              61785.3                             314.8813
359111                              28724.74                            314.8813
399683                              46158.31                            314.8813
471811                              53700.82                            314.8813
488708                              23395.31                            314.8813
499585                              17298.08                            314.8813
516449                              20866.09                            314.8813
533665                              19240.16                            314.8813
596487                              55642.9                             314.8813
681350                              66934.07                            314.8813
701508                              18246.54                            314.8813
```

5. Query Processing
- Join Physical Operators -

* In-memory hash join ($h = \lfloor ID/3 \rfloor$)

6. Processamento de Transações

- O Problema de Concorrência em SBDs -

Entrelaçamento de operações de transações