Motor base de dados:
1. Gestor de armazenamento: gestão de armazenamento, e de acesso a discos e manipulação de ficheiros;
2. Processador de query: processa/interpreta, compila e descobre a melhor forma de obter os dados que o utilizador quer obter;
3. Gestor de transações: permite escrever vários comandos em sequências e garantir que se falhar alguma coisa, volta ao início; permite ter recuperação de dados.
ACID: Atomicidade: todas as operações são executadas, ou nenhuma delas é executada;
Consistência: no fim de 1 transação, quando esta é confirmada, a DB está num estado consistente; Integridade: não há alterações que violem a integridade dos dados; Durabilidade: mesmo que 1 disco avarie/acidente c/ o servidor, se o servidor estiver bem configurado nunca se perdem os dados, vão ser sempre recuperados.

3 tipos de bases de dados:
1. centralizadas: há 1 servidor, c/ vários processos a correr e vários utilizadores que podem aceder a esse servidor
2. paralelas;
3. distribuídas: maior desempenho; ex: DB correm na cloud em vez de 1 servidor, há 1 conjunto de servidores distribuídos e a operação que se faz num servidor é replicada em todos os outros.

Arquiteturas cliente-servidor:
1. arquitetura de 2 níveis: 1 cliente e 1 servidor
2. arquitetura de 3 níveis: 1 cliente, 1 servidor de aplicações e 1 servidor de DB

Componentes ER:
1. DDL: parte que permite definir e criar relações (define tipo de dados, integridade);
2. DML: manipulação dos dados (insert/delete/update)
3. Definição de restrições de integridade
4. Definições de vista DDL (comandos de vista): diferentes utilizadores só podem ver algumas coisas
5. Controlo de transações (COMMIT/ROLLBACK);
6. Autorização: quem pode fazer/aceder algo;
7. Embebida noutras linguagem;

Tipos de linguagem:
1. imperativas: programador diz ao sistema qual o conjunto de operações que deve fazer p/ chegar ao resultado;
2. funcionais: baseadas em funções que tem de ser avaliadas;
3. declarativas: o programador descreve a informação que queremos obter (SQL)

Transações:
Conjunto de operações que tem de ser todas completadas ou todas abortadas, não podem ficar a meio;
1 transação c/ sucesso leva a que 1 DB vá de 1 estado p/ o outro;

Concorrência:
Muitas transações que têm lugar ao mesmo tempo;
Se as transações concorrentes alterem dados diferentes – não há problema;
Se as transações concorrentes quiserem aceder aos mesmo dados – problema;
3 problemas:
1. updates perdidos;
2. ver dados que ainda não foram confirmados;
3. fazer leituras de dados inconsistentes;

Serialização:
Se duas ou mais transações são lançadas em simultâneo tem de ser possível p/ o motor de DB, dizer qual confirmou 1º e depois; se isto não acontecer, o motor de DB deteta e anula; o resultado final tem de ser como se todas as transações fossem executadas em série.

ER: 1. esquema de DB: estrutura de todas as relações que existem na DB
2. instância de DB: é 1 imagem da DB num determinado momento
Chave Primária: identifica univocamente 1 linha numa tabela;
Chave Forasteira: relaciona 1 linha de 1 tabela c/ a linha de outra;
Super Chave: 1 ou mais atributos que podem ser usados univocamente p/ identificar 1 linha numa relação (PK é o conjunto mais pequeno)
Chaves candidatas: podem ser várias; todos os conjuntos de atributos que podiam ser chaves primárias; ÚNICOS

Entidades: onde guardamos os dados;
Relação: relacionamento entre as entidades;
Carnalidade:

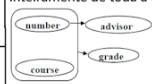
herança 0..1 1..1 0..n 1..n e.f. 0..n e.f. 1..n
Tipos de participação: total, parcial (ver ->)

Regras das dependências funcionais:
1. refletiva: se Y for 1 subconjunto de X, então X determina Y
2. aumento: se X determina Y, então XZ determina YZ p/ qualquer Z
3. transitiva: se X determina Y e Y determina Z, então X determina Z
4. união: se X determina Y e X determina Z, então X também determina Y e Z
5. Decomposição: se X determina Y e Z, então X determina Y e X determina Z separadamente
5. pseudo-transitiva: se X determina Y e YZ determina W, então XZ determina W

Entidade Fraca: só faz sentido no contexto de outra entidade (depende de outra – entidade forte);

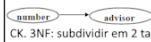
Especialização (Herança): há entidades que se especializam noutras entidades
1. Overlapping: permitido ser várias coisas;
2. Disjunta: só pode ser 1;
3. Total: cada 1 das entidades tem de pertencer a 1 das entidades fracas;
4. Parcial: pode haver entidades que não pertençam a nenhuma entidade fraca;
Há entidades que podem herdar de mais do que 1 entidade – Herança múltipla;

Normalização:
1NF: O domínio dos atributos consiste apenas de valores atómicos. Ñ existe + q 1 atributo a descrever as mesmas características.
2NF: Está na 1NF. Todos os atributos q ñ forem PK dependem inteiramente de toda a chave candidata.



esta tabela NÃO está na 2NF porque adv. e off. dependem só de number. 2NF: subdivir em 2 tabela: [num|course|grade] / [num|adv|office]

3NF: 2NF.Todos os atributos dependem exclusivamente das CKs



esta tabela NÃO está 3NF pq off. depende de adv., que ñ é CK. 3NF: subdividir em 2 tabela: [num|adv] / [adv|office]
Forma Normal de Boyce-Cood: Todos os determinantes têm de ser chaves candidatas;

3 tipos de anomalias (Normalização):
1. inserção: não podemos adicionar dados à DB, porque nos falta informação;
2. update: quando altero 1 registo posso estar a criar inconsistência de dados c/ outros registos;
3.remoção: quando apago dados de 1 registo, perco dados que não queria perder;

Propriedades das transações (ACID):
1. Atomicidade: todas as operações são completas ou então são abortadas, não pode ficar a meio;
2. Consistência: no fim de 1 transação, quando esta é confirmada, a DB está num estado consistente;
3. Isolamento: os dados usados durante 1 transação não podem ser usados noutra transação, enquanto a 1ª não estiver completa, ou seja, quando tem duas transações em simultâneo elas não vêm as alterações 1 da outra até que estejam confirmadas;
4. Durabilidade: a partir do momento que fizer o COMMIT de 1 transação esta não pode ser anulada (ROLLBACK);

Gestão de transações SQL:
1. read only: transação só de leitura, se tentar fazer update dá erro e é igual ter commit ou rollback 1 vez que não vamos alterar nada;
2. autocommit: deve estar desligado, uma vez que não dá p/ dar rollback, pode ficar num estado inconstente;
3. savepoint: permite marcar pontos p/ os quais posso voltar atrás
- savepoint <my_savepoint> (guardar)
- rollback to < my_savepoint > (apaga)
- realease < my_savepoint > (destrói)
4. transaction log: guarda todas as operações; usado p/ fazer recuperações;

Escalonamento de transações:
É 1 mecanismo que estabelece a ordem dos comandos; Os mecanismos de bloqueio são quem permite estabelecer esta ordem e p/ determinar a ordem correta o escalonador tem diferentes algoritmos.
Nem todas as transações são serializáveis. Há transações que o motor de DB não consegue decidir a ordem de execução; neste caso faz ROLLBACK e avisa o programador que 1 delas tem de ser cancelada.

Controlo de concorrência:
Ter mecanismos de bloqueio que permitem evitar alterações concorrentes aos dados, p/ não tenhamos inconsistências; Estes mecanismos garantem a serialização das transações; Suportam a propriedade de isolamento, ou seja, não altera dados antes que os outros confirmem ou anulem as alterações que fizeram.
1. Perda de updates: 1 das operações realizadas foi perdida, porque foi escrita por cima (overwritten) por outra operação;
2. Leitura de dados não confirmados: acontece quando temos 2 transações e não existe 1 mecanismo de isolamento, ou seja, a transação 1 consegue ler dados que não foram confirmados pela outra transação;
3. Informação inconsistente: 1 comando numa determinada transação acede a dados antes e depois de ter sido feita alguma transação (antes da transação acabar);

Conflitos entre shared/exclusive:
Lock pode ter 3 estados:
1. Desbloqueado
2. Shared (read)
3. Exclusive (write)

1 lock partilhado impede que outra transação obtenha 1 lock exclusivo.

Mecanismos de bloqueio:
Os locks podem ser feitos:
1. implicitamente: alteração de dados e a DB bloqueia (UPDATE);
2. explicitamente: lock table, select for update;

Modos de bloqueio:
1. Bloqueio binário: (1 ou 0)
Ou está bloqueado ou desbloqueado (Problema: restrito);
Nenhuma transação pode usar 1 objeto que esteja bloqueado por outra transação; No início de 1 transação devem bloquear o objeto e desbloquear no fim; Gerido automaticamente pela DB;
2. Share/exclusive lock:
Permite ter operações de leitura em paralelo c/outras operações de leitura e em paralelo c/ operações de escrita, mas nunca operações de escrita c/escrita;
Lock partilhado: existe quando é possível que tranqsações concorrentes possam ler o mesmo objeto todas podem aceder ao objeto p/ leitura (quando fazemos 1 select); **LER**
Lock exclusive: acesso reservado p/ 1 transação de escrita que vai escrever sobre 1 objeto e não quer que mais ninguém tenha acesso (quando fazemos 1 update é criado 1 lock exclusivo sobre os registos que são alterados; se fizermos lock table é sobre toda a tabela); **ESCREVER**

LOCK TABLE tem estar antes do SELECT, porque ao consultar 1º pode ter lido e quando bloqueia já não vê a mesma informação.

Bloqueios implícitos:
1.insert/update/delete: row exclusive
2. select: acess share
3. alter table: share update exclusive

Modos de bloqueio de tabelas SQL - 8:
1. acess exclusive: mais ninguém pode ler nem mexer na tabela;
2. exclusive: apenas leituras na tabela podem ser feitas em paralelo; não pode ser bloqueado por outra tabela (permitido acess share);
3. share: protege contra alterações concorrentes; não podem alterar nem colocar em lock exclusive; podem ler mas só quem faz share pode alterar (as alterações só são vistas após o COMMIT);
4. acess share: (menos restrito) impede que outras transações adquiram o modo acess exclusive;
Nota: select... for update-> bloqueia

Métodos de controlo de concorrência – Locking:
Tem 1 granularidade que indica o nível de bloqueio que quero implementar;
Existem 5 níveis de bloqueio:
1. Database – Bloquear a DB toda, só 1 transaçao é que pode usar a DB naquele momento;
Problema: enquanto está ativo ninguém faz nada;
2. Table – quando queremos fazer alterações na estrutura da tabela; quando estou a fazer operações sobre os dados de 1 tabela e quero ter a certeza que ninguém altera aquela tabela, mas outras podem ser alteradas;
3. Page (ler conteúdo do disco) – page = página de memoria -> quando quer bloquear 1 bloco de registos, todos ficam bloqueados ao mesmo tempo;
4. Row – 2 transações a tentar alterar os mesmos dados 1 delas bloqueia (mesmos dados na mesma linha);
Nota: ter atenção a quando quer mudar a mesma variável; se muda o nome e posteriormente o nº e estes são comuns há problema;
Nota: quando da update a 1 linha que não existe: "zero rows updated";
5. Field (atribute) – por coluna e por linha (campo) "bloquear o nome tem registo no campo 20";
Problema: muito controlo, overhead substancial;

qd criar índices: boas ideias:
ats. c/ mts queries (ex: PKs),
ats. UNIQUE, FKs (juntar tabelas)
ats. ordenados

Multitable Clustering:
1. Guardar os registos de 2 tabelas próximos 1 do outro; É útil em termos de desempenho; Leva a acessos mais eficientes para alguns tipos de queries;
2. Chave do cluster diz com é que as tabelas vão ser juntas e estrutura os dados no armazenamento;
3. Problemas de desempenho quando quero apenas aceder a parte desses dados;
4. Dependendo do tipo de queries, usamos ou não clustering.

3 tipos de leituras [ISOLAMENTO]:
1. dirty read: ler dados que ainda não foram confirmados;
2. nonrepetable read: lê 1 linha no momento T1 e no T2 lê a mesma linha e obtém resultados diferentes;
3. phanton read: 1 transação executa 1 query num determinado momento e quando executa essa mesma query mais à frente obtém linhas adicionais que não estavam na execução no 1º momento;

Isolamento: Nível de proteção e o que conseguem ver umas das outras;
1. read uncommited: Permite ler dados que não foram confirmados pelas outras transações; não é preciso qualquer tipo de lock; problema de consistência de dados;
2. read commit (default): Lêem apenas dados que já foram confirmados (COMMITED);
3. repeteable read: Garante que as queries devolvem sempre resultados consistentes; (dados commited antes da trans. começar)
4. serializável: Mais restritivo; Quer se garantir que a execução das transações segue, dá o resultado como se elas fossem executadas em série. Impede dirty reads, nonrepetable read e phanton read.

Problemas de bloqueio:
1. Deadlocks:
Acontecem quando duas ou mais transações esperam indefinidamente umas pelas outras; Não existem deadlocks quando existem apenas shared lock (só leituras não permitem levar a deadlocks);
Resolver: Ter a ordem certa nas operações que constam nas várias transações; constrói 1 wait-for-graph; p/ resolver 1 deadlock libertar 1 das transações;
2. O que resulta das transações não seja serializável:
1 conjunto de transações é serializável se o resultado final for igual à execução sequencial das transações;
Resolver: Two-Phase-Locking:
1ª fase: [crescimento] obter todos os loks que preciso; fazer todas as alterações;
2ª fase: [encolhimento] libertar todos os lock que usei; no final garante que todas as transações são serializáveis.

Índices:
Estrutura de acesso auxiliar que permite otimizar o desempenho. Índices de B-tree permitem aceder de forma eficiente aos registos que estão na tabela da DB; A árvore balanceada [Todos os ramos têm a mesma profundidade]
Indexar as colunas que são usadas mais vezes nas condições de pesquisa
1. Multiple Single-Key: 1 índice;
2. Multiple Keys: Vários índices;
3. Covering: índices sobre múltiplas colunas que não são usadas como chaves de pesquisa

Armazenamento físico:
Aceleradores de I/O em vez de discos magnéticos; Usar RAID (vários discos em paralelo, discos replicados);
Contenção de discos; Minimizar conflitos entre tabelas; Separar os ficheiros de dados em discos diferentes;
Aceder aos dados em paralelo (melhor desempenho);

Performance Tuning: Otimizador todo o sistema de forma a garantir que as respostas aos utilizadores são o mais rápidas possível;
Desempenho:
1. Tempo de resposta (menos possível);
2. Throughput (nº de operações executadas por unidade de tempo);

Níveis de isolamento: permitem dizer que tipos de operações são possíveis em cada transação;
Servem p/:
1. fazer relatórios usando os dados que têm no momento;
2. os diferentes níveis têm diferentes restrições
+ bloqueio + restrições - concorrência

Aplicação de base de dados (3 níveis):
1. front end [interface c/ o utilizador];
2. servidor de aplicações [corre a lógica de negócio];
3. base de dados [tabelas e dados]

Vistas [baseadas no SELECT]:
1. Simples: baseadas numa única tabela e não tem funções de grupo;
2. Complexas: acesso a várias tabelas, posso incluir funções e GROUP BY;
Quando 1 vista é criada, é criada 1 tabela temporária c/ o SELECT dessa vista; As vistas servem p/ restringir o acesso aos dados; Simplificam o acesso aos dados; Permitem que os mesmos dados possam ser vistos de maneiras diferentes p/ outros utilizadores; O comando CREATE não executa o SELECT apenas o guarda no dicionário de dados; Apenas quando acedo à lista é que o SELECT é executado; É possível dar INSERT/UPDATE/DELETE das tabelas através das vistas; P/ manter a integridade das vistas – WITH CHECK OPTION, tem de satisfazer a condição do WHERE; Apenas dá p/ fazer DELETE em vistas simples; As vistas tem de ser apagadas.

Interação c/ utlizador:
1. Aplicação cliente (utilizador) gera a query;
2. Query é enviada pelo motor de DB;
3. O motor de DB executa a query;
4. Devolve os resultados ao cliente;

Sort cache:
Área de memória partilhada usada para fazer ordenamento [ORDER BY] e agrupamentos [GROUP BY]

Processamento de 1 query:
1. parsing: processar a query e escolher o plano de execução + eficiente;
2. execution: executa a query e faz os passos do plano de execução 1 de cada vez;
3. fetching: vai buscar os dados para fazer cada um dos passos;

Registos:
1. Tamanho fixo [char/data/numeric]: Todos os registos têm o mesmo tamanho; Encher até estar cheio;
Problema:
A) Ter registos que vão estão entre 2 blocos;
B) Apagar 1 registo; Solução: Lista ligada;
2. Tamanho variável [varchar/ long] – 2 formas de representar:
A) Representar 1 registo: 1º tamanho fixo, 2º tamanho variável; no meio o bitmap [nulos];
B) Guardar cada registo no determinado bloco: Organizar registos dentro de 1 bloco; Header – informação de controlo, guardar ponteiros p/ registos e espaços livres;
Problemas:
1. Inserir: procurar espaço vazio; 2. Eliminar: mover todos os registos e mudar o ponteiro de espaço livre; 3. Quando o tamanho varia: andar p/ trás/frente, caso não haja espaço mover p/ outro bloco; solução: deixar margem;

Objetos grandes [guardar]:
Blob: binary long object
Clob: character long object
1. guardar na DB c/ todas as restrições (backup e eficiência);
2. ponteiro p/ o ficheiro que está no file system [problema: integridade referencial quando o ficheiro é alterado]

Partionamento:
Dividir 1 tabela em partes mais pequenas – armazenar separadas (posso guardar em discos diferentes);
Há 1 chave que diz como dividir;
Útil em tabelas grandes;

Organização de ficheiros:
DB é mapeada segundo 1 conjunto de ficheiros que estão no disco; Cada ficheiro está organizado como 1 sequência de registos [registos das tabelas]; Cada ficheiro está particionado num conjunto de blocos de tamanho fixo; 1 ficheiro é composto 1 conjunto de blocos; 1 bloco de dados contém 1 conjunto de registos; Pode haver registos de tamanho fixo e tamanho variável;

Restrições de integridade:
1. Chave primária/entidade: cada tabela tem 1 chave primária e o valor dessa PK nunca se repete;
2. Domínio: definimos quando criamos 1 tabela, ex: quando criamos o atributo nome tem 1 tipo e 1 tamanho;
3. Referencial: existência de 1 chave forasteira;
4. Complexas: tem de ser codificadas; não são a nível das tabelas (TRIGGERS);

Perspetiva utilizador:
Processos estão a correr – cash (memória);
Estruturas físicas – o que está nos ficheiros;

Otimizador baseado em:
1. regras: custo fixo de cada operação;
2. custos: estatísticas que recolhe;

Triggers: Algo que se ativa quando acontece alguma coisa; Define 1 ação que deve ser executada quando outro evento acontecer;

Procedimentos: alterar tabelas/alterar coisas na DB;
Funções: devolver cálculos/operações que computem;

Vulnerabilidade [SQL injection]: alguém injeta código SQL que modifica a estrutura da DB; - não há validação de inputs;

Organização dos registos dos ficheiros:
1. Heap file organization: p/ qualquer bloco dentro do ficheiro (onde houver espaço);
2. Sequencial: organizados por 1 chave sequencial (aceder melhor/ inserir pior);
3. Clustering de tabelas: intercalar registos de 1 tabela c/ os de outra e guardá-los juntos;
4. B-tree [Hashing]: a organização é dada por 1 função de hash;

PL/pgSQL: podemos agrupar e criar blocos de computação dentro do motor de DB;
Vantagem: não há comunicação na rede, logo melhor desempenho; Baseia-se em blocos e pode ter sub blocos de código dentro desse bloco; Anónimo: código todo; Não anónimo: invocar [CALL];

tablespace: local no disco onde o sistema de gerenciamento de banco de dados armazena os dados físicos para as tabelas e índices. extent: bloco contíguo de dados no disco alocado para tabs. e índs. permite gerir + eficiente o espaço do disco.
database: conjunto de dados relacionados que são armazenados juntos e gerenciados como uma unidade pelo DBMS.
schema: container lógico para os objetos da db, como tabelas, índices, views e procedimentos armazenados. usado para organizar os objetos em grupos para facilitar o gerenciamento e a segurança.

Arquitetura da base de dados:
Tamanhos típicos 4kb/8kb/16 kb
1. Ficheiro de dados: Onde o motor de DB vai guardar as tabelas; Guardar índices e vistas; Cada 1 dos ficheiros pode estar replicado; Extents – quando o ficheiro está cheio, o motor da DB aloca mais espaço;
2. Table Space: conjunto de ficheiros; Permite-me facilitar a gestão dos ficheiros;
3. Data Cache [Buffer Cache]: Onde o motor da DB vai guardar dados que está a processar; Os dados são lidos dos discos p/ a data cache; O tamanho é configurável (o admin pode definir); Problema: ter 1 data cache muito grande: problema na recuperação dos dados; Garantir que é grande o suficiente p/ conseguir responder ao máximo de pedidos dos clientes; Partilhada por todos os utilizadores;
LOG – escrito imediatamente no disco [comandos]; *1
4. SQL Chache [Cache de procedimentos]: Guardar os comandos SQL que fomos executando (+ recentes) e o plano de execução;
5. Listener Process: Processo que está à escuta de novas ligações; Recebe ligações e comandos do cliente;
6. User Process: Depende do número de ligações; Processa os comandos recebidos pelo cliente; O nº máximo de user process é definido pelo admin;
7. Scheduler e lock manager: organiza a execução concorrente dos comandos SQL; Garante que as transações são serializáveis; Colocar locks nos objetos [tabelas de registos];
8. Optimizer Process: Pega nos comandos que submetemos e encontra a forma mais eficiente p/ responder;

Estatísticas da DB:
O motor de DB recolhe estatísticas sobre tudo, permite que o motor de DB tome as melhores decisões; podem ser geradas:
1. dinamicamente: o motor de DB encarrega-se de ir gerando estatísticas ao longo do tempo;
2. manualmente: o admin da DB pode ir gerar/atualizar as estatísticas;

Performance (Cliente/Servidor):
Escolher 1 boa configuração de DB; Usar índices; Distribuir os ficheiros por vários discos, otimiza o desempenho;
Servidor – configurar da melhor forma;
Cliente – otimizar o SQL;

Indexação: otimizar acesso aos dados:
1. Funtion based index/hash índex: base em funções de hash;
2. bit map: array de 0/1 que representam a existência de 1 condição;
3. B-tree;

Plano de execução: Define quais as operações que vão ser executadas;
1. full table scan: percorrer a tabela toda;
2. table acess (row iD);
3. index scan (range): usar 1 range do índice;
4. index acess (unique): está a usar 1 indice numa coluna c/ valores únicos;
5. nasted loop: processar várias tabelas ao mesmo tempo e p/ cada linha de 1 processa de dados da outra tabela;

Hash index:
Aplicamos 1 função de hash sobre 1 coluna da tabela e c/ base nessa função a coluna é organizada e existem buckets (índices) que tem ponteiros p/ as tabelas; Melhor p/ colunas c/ muita variedade como o nome, não seriável c/estados civis

Segurança:
Valores (dados)/Ameaças; Permite regular o acesso aos dados e dar autorizações explicitas aos dados; Gerir os riscos
3 propriedades: (CIA)
1. confidencialidade: os dados(não são recuperáveis) têm de ser confidenciais
2. integridade: alterados p/ quem tem permissão
3. disponibilidade: os dados estão sempre disponíveis quando necessários
P/ garantir as propriedades:
1. Autenticação
2. Autorização (controlo de acessos)
3. Auditoria (monitorizar as operações de cd utilizador)
4. Ação (reagir* contra potenciais problemas)

Arquitetura DW:
Sistema fonte: sistemas transacionais; operações do dia a dia
Area de estagio: guardar periodicamente os dados p/ depois guardar da dw [extrair] -> limpar/transformar/dar consistência/preparar [carregamento]
Tranformaçao: remover campos que não vao ser uteis na tomada de decisão; combinação de vários dados; geral PK (surrugate key; construir outra estuturas

OLTP-> DB transacionais
Interface dos utilizadores: convertem as pesquisas em sql; queries ad-hoc
Slice e Dice: partir em diferentes pedaços p/ tomar decisões, partir aos bocados p/entender melhor os dados
Drill-down & role-up: pesquisa genérica (geral)-> aprofundar a pesquisa detalhe)

OLAP: Processamento Analitico de Dados: pesquisa e apresentação de DW
ROLAP (modelo relacional desnormalizado)
MOLAP(cubos)

Segurança dos utilizadores:
1. Garantir a segurança da pass;
2. Forçar mudança periódica;
3. Gestão de privilégios;
4. Definir diferentes esquemas de segurança

Administraçao da Base de Dados:
Administrador: responsáveis por planear, organizar, controlar e monotorizar -> tudo o que tem a ver c/a DB
1. Gestão: foca no gestão das pessoas e as interaçoes c/ os utilizadores;

Segurança do sistema: Gestão dos utilizadores; Privilégios; Autenticação; Segurança a nível do sistema operativo;
Segurança dos dados: Controlar o acesso às tabelas; Privilégios; Definir roles; Segurança deve ser discreta!
Segurança dos utilizadores: Todos os utilizadores devem: Garantir a segurança da password; forçar mudança periódica; gestão dos privilégios; definir diferentes esquemas de segurança (end users que fazem uso da parte do cliente; application developers que criam as tabelas; admins);
Auditoria: define políticas e processos; registo de tudo o que os utilizadores fizeram; não ter a auditoria sempre ligada o que leva a problemas de performance.
Roles: Privilégios; Atribuir a utilizador; Reduzir e simplificar as tarefas de admin
Privilégios: 1 role tem vários privilégios; direitos de executar/aceder;
2 tipos: 1. Sistema – que ações é que 1 utilizador pode fazer; 2. Objetos – que objetos é que o utilizador pode aceder (sobre tabelas de outros);
GRANT: dar; REVOKE: retirar; Se criar 1 tabela tenho sempre privilégios sobre esta.
Perfis de utilização: Limitar os recursos (tempo de cpu e sessão, espaço de memória) usados por cada utilizador (1 utilizador só pode usar x memória);

Query – otimização (SQL):
1. Modo de operação: a) automático: o motor de DB está encarregue de encontrar o caminho mais eficiente de executar (comum); b) manual: o programador é que define os passos que a DB deve fazer p/ executar a query;
2. Tempo: a) estático: plano de execução é definido no momento da compilação da query; b) dinâmico: o plano de execução é definido no momento da execução;
3. Informação usada: a) estatísticas: o motor de DB vai usar a informação da DB naquele momento p/ saber qual é o melhor plano de execução; b) regras;

Bitmap index:
Associado a cada linha da tabela 1 bit que diz se o valor existe ou não; Melhor p/ colunas sem muita variedade, por exemplo estado civil.

Índices são bons e otimizam o desempenho mas se tiver muitas operações de inserção e apagamento eles atrasam o processo, porque sempre que se altera 1 registo tem de se atualizar o índice.

Otimização:
Hardware [quantos mais recurso tiver melhor]: capacidade de processamento (CPU); memória disponível; throughput (input/output);
Abordagem holística: otimizar os vários fatores;

Tarefas de administração:
Storage:
1. Tablespace: Definir localizações no sistema de ficheiros; controlar o layout do disco; definir em que tablespace fica cada tabela
2. Database: dentro do mesmo sistema tenho varias DB; privilégios de acessos p/ cada DB

Limpeza dos dados: resolver erros; corrigir inconsistências; remover duplicados(atenção M e m); preencher dados em falta

Key value storage systems: não estruturas nem interpretados (conjunto de bytes); tabelas; json; Limitações: só 3 operaçoes (put/get/delete); não tem transações;

Vulnerabilidades:
SQL injection/password fácil;
1. Técnicas – no código/web;
2. Gestão – organização que não educa os utilizadores;
3. Cultural – não esconderem a password;
4. Procedimentos – forçar passwords complexas;

B-tree index:
Árvore balanceada, ou seja, a mesma profundidade, cada nó tem vários filhos. Cada nó tem de ter pelo menos n/2 preenchidos, pelo menos 2 filhos ou mais. Menores esquerda, maiores direita (alf). P/ cada nó -> ponteiro p/ a tabela.
Quando temos 1 update, temos de mudar tudo p/ garantir que a arvore está balanceada-> separar nos/juntar nos (inferior a n/2).
Inserção: Posição onde deveria aparecer;
Inserir o registo; Se o número de ponteiro for maior que n temos de dividir (nó da folha != nó não folha);

Quem pode aceder às tabelas: Privilégios! (de acordo c/ as necessidades das operações->política de segurança);
Políticas de segurança: políticas e normas que garantam 1 acesso seguro ao sistema e que garantam que dê p/ fazer auditoria
4 níveis: 1. Segurança do sistema; 2. Segurança dos dados; 3. Segurança dos utilizadores; 4. Auditoria;

Dados operacionais: atividades operacionais; acessos escrita/leitura; transacoes predefinidas; acessos p/ 1 conjunto reduzido de registos; informação atualizada em tempo real;
Dados DW: registos históricos; apenas queries; ad-hoc queries e relatorio; muitos registos em cada acesso; carregamentos periodicos

	Relations	NoSQL
Storage Model	Relations	KeyValue
Schemas	Static	Dynamic
Scaling	Difficult	Horizontal
Transactions	Yes!	Managed by app
Data Manipulation	SQL	OO APIs

Tipos diferentes de DB:
Data Warehouse – apoio à decisão; DB grandes; Tomar decisões; Dados redundantes e raramente tem alterações; Construídas a partir das DB transacionais
Caraterísticas principais:
1. Dependência Temporal (tipo histórico, o que aconteceu ao longo do tempo p/ tomar decisões)
2. Não voláteis: Não há remoções; o carregamento de dados é periódico; acumulação de dados [comparar o atual c/ o anterior]; dados muito velhos (não ter em conta)
3. Orientadas p/ 1 determinado objetivo
4. Dados de diferentes fontes e têm de ser consistentes (a mesma pessoa tem nomes diferentes em diferentes sistemas)
5. Desenhados p/ queries de pesquisa [podem demorar horas]
6. Guardam grandes volumes de dados

Modelo Multidimensional:
1. forma desnormalizada
2. orientada aos utilizadores (pesquisas)
3. modelo de dados é intuitivo!
4. Modelo Estrela: Tabela de Factos (95% -> Há várias); Dimensões (carateriza os factos)

R I P

Schemas:
Há vários esquemas; cada esquema tem 1 dono; Por defeito estamos no esquema public!

Backup e recuperação: dados tem de ser recuperados; garantir que não se perde nada; gestão de desastres; devem ser feitos backups periódicos (full, incremental - só o que mudou, concorrente- em paralelo c/ o que acontece na DB)
Tecnicos: avaliar, selecionar e instalar a DB; suporte ao desenho da DB; manutenção do dia a dia

Carregamento: pegar no que transformei e colocar no local certo da DW
Sistema de Ficheiros distribuído: os dados são guardados num conjunto de maquinas mas vemos com 1 único; muito grandes; redundância (maquinas baratas mas servem p/guardar dados)
Hadoop: só temos 1 nome (namespace); cada ficheiro é partido por cada bloco [data nodes]
Sharding: armazenar em múltiplas DB; 1query ser executada em varias DB; chave de sharding

Limitaçoes das DB relacionais: as juncoes de dados são dispendiosas; é difícil escalar horizontalmente (é difícil adicionar + recursos e tirar partido desses recursos); caras;
-desempenho
-disponibilidade
-tolerancia ao particionamento

Paralelas e distribuídas: replicações (caso 1 maquina falhe); distribui por varias maquinas
1 replicacao e consistência [CAP]: availibility- redundância (case 1 maquina falhe); consistência- a replica A deve ter o mesmo que a B (todas tem a mesma versão); particionamento da rede-1 conjunto de maquinas que comunicao pela mesma rede (não garantimos as coisas acima)

Exemplo de Procedimento:
```
create or replace procedure ex3()
language plpgsql
as $$
declare
   livros_infor cursor for
   SELECT autores.id_autor,  autores.nome
   from autores;
   max_livros numeric;
   max_livros_genero numeric;
begin
   for a in livros_infor
   loop
      select count(*) into max_livros from
livros where livros.id_autor =a.id_autor;
      Select max(count) into
max_livros_genero
      from (select count(*) from livros
      where livros.id_autor = a.id_autor group
by livros.genero) as a where count is not
null;

      insert into temp(col1, col2, message)
      values(max_livros, max_livros_genero,
split_part(a.nome,' ',  1));
   end loop;
end;
$$;
```

```
CREATE OR REPLACE FUNCTION
search_auctions(keyword VARCHAR(512))
RETURNS TABLE(
   auct_id INTEGER,
   descrip VARCHAR(512)
)
AS $$
BEGIN
   IF keyword~E'^\\d+$' THEN
         return QUERY SELECT id, description FROM
auction
            where (description like ('% ' || keyword || '
%') or  auction.EAN_ISBN = (keyword::INTEGER))
            ORDER BY end_time DESC, id ASC;
   ELSE
      RETURN QUERY  SELECT id, description FROM
auction
            where (UPPER(description) like ('% ' ||
UPPER(keyword) || ' %')
            or UPPER(description) like ('% ' ||
UPPER(keyword) || '%')
            or UPPER(description) like ('%' ||
UPPER(keyword) || ' %'))
            ORDER BY end_time DESC, id ASC;
   END IF;
END;
$$
LANGUAGE plpgsql;
```

Dicionário de dados: metadados, dados sobre dados

Strict: tem de encontrar qualquer coisa, se não encontrar gera 1 excessão [NO DATA FOUND]

CRUD: conjunto básico de operações que podemos fazer sobre 1 elemento da DB – CREATE/READ/UPDATE/DELETE

Exemplo de Função c/ trigger:
```
CREATE OR REPLACE
FUNCTION bid_notification()
RETURNS trigger AS
$$
DECLARE
   cur cursor for
   select DISTINCT bidder_id as "reciever" From bidding
where
   auction_id = NEW.auction_id
   and bidder_id <> NEW.bidder_id;

   new_bidder users.username%type;
   new_value bidding.price%type;
   auction_title auction.title%type;
   auction_id auction.id%type;
   mess VARCHAR;
BEGIN
   SELECT username into new_bidder from users where
id = NEW.bidder_id;
   SELECT price into new_value from bidding where id =
NEW.id;
   SELECT title into auction_title from auction where id =
NEW.auction_id;
   SELECT id into auction_id from auction where id =
NEW.auction_id;
   for elem in cur
   loop
      mess := concat('Licitacao ultrapassada no leilão ',
auction_id, '-> ', auction_title , ' -> pelo utilizador ' ||
new_bidder || ' no valor de ' || new_value || '€.');
      insert into notifications (message, mes_time,
users_id)
      values(mess, current_timestamp(0) , elem.reciever);
   end loop;
   RETURN NULL;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER onBidding
AFTER INSERT ON bidding
FOR EACH ROW Execute PROCEDURE bid_notification();
```

Drop Table: apaga a tabela
Delete: apaga o conteúdo da tabela

```
create or replace procedure f10_ex6(p_livro
livros.id_livro%type)
language plpgsql
as $$
declare
   v_id_livro livros.id_livro%type;
   v_id_autor livros.id_autor%type;
begin
   select id_livro,id_autor
   into strict v_id_livro,v_id_autor
   from livros
   where id_livro=p_livro;
   if v_id_autor=17 then
      insert into autores
(id_autor,nome,morada,sexo,nacionalidade,gene
ro)
      values(80,'Luís Moreno
Campos','Lisboa','M','Portuguesa', 'Informática');
      update livros set id_autor=80 where
id_livro=v_id_livro;
   end if;

exception
   when unique_violation then
      insert into erros(sql_state, message, data)
      values(sqlstate,sqlerrm,current_date);
      update livros set id_autor=80 where
id_livro=v_id_livro;
   when no_data_found then
      insert into erros(sql_state, message, data)
      values(sqlstate,sqlerrm,current_date);
   when others then
      insert into erros(sql_state, message, data)
      values(sqlstate,sqlerrm,current_date);
end;
```

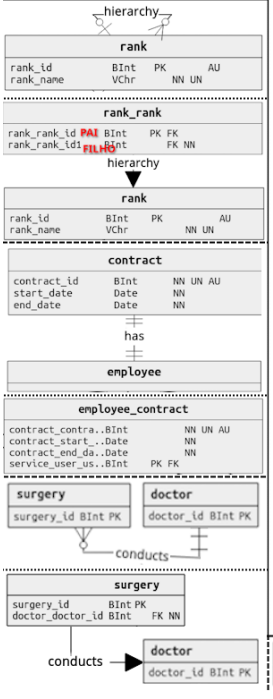
coalesce round trunc ceil floor
union, u all, intersect, except
any all
distinct
concat upper lower initcap
substr(s, p, [n])
max min sum avg count
length string_agg(n, ',')
date_part('year', date)
to_char(DATE '2005-01-01', 'Day')

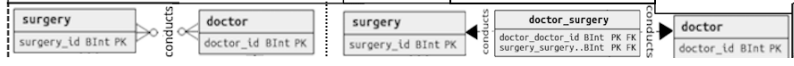
```
SELECT * FROM (SELECT
cod_prod,
  SUM(case when extract('YEAR'
from data_enc) = 2020 then
quantidade else 0 end) as
quant_2020,
  SUM(case when extract('YEAR'
from data_enc) = 2019 then
quantidade else 0 end) as
quant_2019
FROM clientes, encomendas,
itens
WHERE clientes.code_cliente =
encomendas.clientes_code_clien
te
AND encomendas.num_enc =
itens.encomendas_num_enc
AND clientes.localidade =
'Coimbra'
GROUP BY cod_prod
ORDER BY cod_prod) as t where
t.quant_2020 > t.quant_2019;
```

É a DB, e não a aplicação cliente que determina a melhor forma de executar as tarefas;

```
CREATE OR REPLACE PROCEDURE bonus()
language plpgsql [as $$]
DECLARE
   codeCliente clientes.code_cliente%type;
   c1 cursor for
      select code_cliente, Sum(itens.quantidade *
itens.preco) as price
      from clientes,
         encomendas,
         itens
      where clientes.code_cliente =
encomendas.clientes_code_cliente
         AND encomendas.num_enc =
itens.encomendas_num_enc
      group by code_cliente
      order by price desc
      limit 3;
BEGIN
   open c1;
   loop
      fetch c1 into codeCliente;
      exit when not found;
      update clientes set saldo = saldo + 50 where
code_cliente = codeCliente;
   end loop;
   close c1;   END;   $$;
```

```
SELECT num_enc as "Codigo da Encomenda", data_enc as "Data Encomenda",
SUM(quantidade) as "Quantidade"
FROM encomendas, itens
WHERE encomendas.num_enc = itens.encomendas_num_enc
AND cod_prod not in (select cod_prod from itens, encomendas
 WHERE itens.encomendas_num_enc = encomendas.num_enc and extract('YEAR' from
data_enc) = 2020)
GROUP BY num_enc
HAVING SUM(quantidade) > (SELECT AVG(quantidade) FROM itens);
```





*1 -> Log buffer: armazenar transações antes de serem gravadas no log de transações no disco, permite que várias transações sejam gravadas no disco de uma vez, o que pode melhorar o desempenho. (flushing