Base de Dados

A partir de: apontamentos das aulas teóricas, *Database Systems - A Practical Approach to Design, Implementation, and Management* e *SQL - Structured Query Language (14ª Edição) de Luís Damas*;

Capitulo I - Base de Dados

Uma **Base de Dados** é uma coleção de dados partilhados interrelacionados e usados para múltiplos objetivos. Num **sistema de ficheiros**, cada aplicação cria e mantém os ficheiros com os dados necessários para a sua execução. Problemas: alto nível de redundância, inconsistência de informaçãoinflexibilidade, acessos concorrentes, isolamento e integridade dos dados e e elevadois custos de manutenção.

Um sistema de base de dados pretende que um programa possa ser modificado, sem que isso implique alterações nos restantes programas que usam os mesmos dados, baixando os custos de manutenção. Os dados são integrados num **sistema de gestão de base de dados**: conjunto de programas que permitem armazenar e manipular dados, fornecendo-os aos programadores e utilizadores finais tal como são pedidos. O sistema de gestão de base de dados reserva para si os privilégios de acesso físico às bases de dados, sendo quase independente do sistema operativo, tal como os dados consultados e alterados.

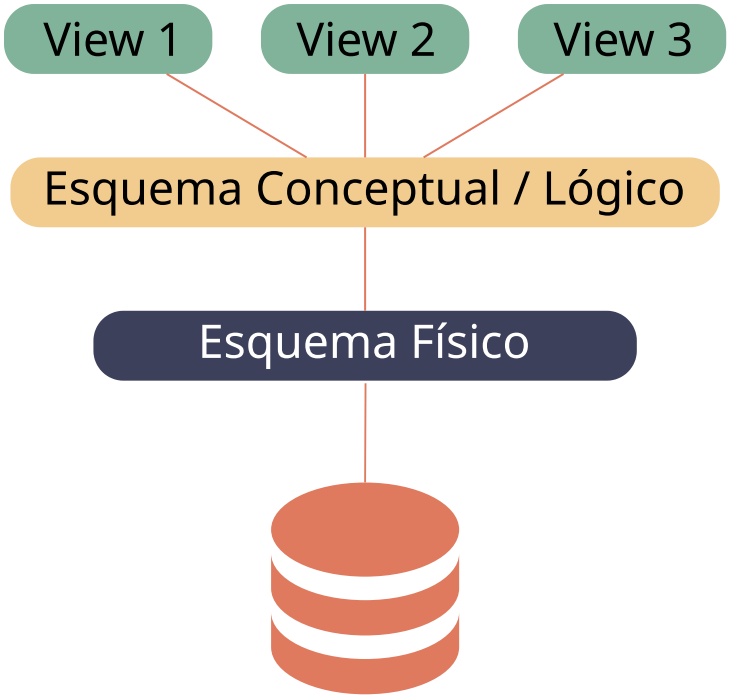
Níveis de abstração:

- *Interno* (menor) - rerepresentação fisica da base de dados no computador. Este nível descreve **como** os dados são armazenados na base de dados, otimizando o desempenhbo e o armazenamento;
- *Conceptual* (intermédio) - Visão comunitária da base de dados, Descreve quais dados são armazenados e as relações entre sí. Contém toda a estrutura lógica independentemente das restrições de armazenamento;
- *Externo* (maior) - Visão individual (de cada utrilizador) da base de dados. Descreve qual parte da base de dados é relevante para um determinado grupo de utilizadores;

Independência dos dados:

- *Física* - A alteraçãp de as+etos relativos à implementação fisica da base de dados, sem que se altere o seu esquema conceptual (mantendo os dados e as suas associações inalterados);
- *Lógica* - Quando é necessária a adição ou a remoção de atributos de uma entidade já existente ou até mesmo criando entidades novas sem alterar a visão do utilizador;

Arquitetura ANSI-SPARC (*American National Standards Institute - Standards Planning And Requirements Committee*):



Considere a arquitetura ANSI-SPARC dum SGBD e os seus níveis de abstração (externo, conceptual e interno). Explique o que se entende por independência lógica dos dados e por independência física dos dados?

Resposta:

- *Independência lógica de dados:*
  - Invariância dos subesquemas externos face a alterações no esquema conceptual;
  - Será possível, na generalidade dos casos, alterar o esquema conceptual sem ter de alterar também o esquema externo. Ou seja, alterações no nível conceptual não interferem, de forma obrigatória, com as "vistas" estabelecidas no nível externo (a menos que haja eliminação de componentes no esquema conceptual, caso em que algumas das "vistas" estabelecidas no esquema externo poderão ser afetadas).
- *Independência fisica de dados:*
  - Capacidade de alterar o esquema interno sem ter de alterar o esquema conceptual. Ou seja, alterações no nível interno não se repercutem no nível conceptual.
  - Isola o utilizador das açterações no esquema fisico dos dados

Um sistema de gestão de base de dados é um sistema de gestão e armazenamento de dados, capaz de aceder eficientemente a grandes quantidades de dados. Serve de intermediário entre o servidor e o cliente, sendo a única entidade responsável pela segurança, integridade e validade dos dados armazenados. Possui:

- Suportes para modelos de dados e linguagens de alto nível:
  - *Data Definition Language* (DDL) - define a estrutura da base de dados e a informação a armazenar;

    - *Data Manipulation Language* (DML, *query language*) - obtém, armazena, altera ou elimina informação da base de dados, podendo ser autónoma ou integrada numa linguagem hospedeira de alto nível;

- Gestão de transações - conjuntos de operações perfeitamente delimitadas em que garantidamente são executadas todas as instruções ou então nenhuma produzirá efeito sobre a base de dados, seguindo as propriedades ACID:

    - *Atomicidade* - as instruções cliente-servidor são indivisíveis (ou são todas executadas (`COMMIT`) ou nenhuma é executada (`ROLLBACK`));
    - *Consistência* - após uma transação, a base de dados deve manter-se consistente;
    - *Isolamento* - devem evitar-se interferências múltiplas, dando a ilusão de que cada instrução é a única a ser executada num dado momento;
    - *Durabilidade* (Persistência) - após um `COMMIT`, os seus efeitos são persistentes/não-voláteis na base de dados e só podem ser desfeitos/alterados por outras transações;

- Controlo de acesso:

    - *Segurança* - É um dos requisitos básicos exigidos a um SGBD. Consiste em proteger os dados armazenados dos acessos não autorizados e garantir que todas as operações executadas sobre a base de dados são executadas por utilizadores (aplicações) devidamente credenciados.
    - *Integridade* - Por definição, uma base de dados está num estado de integridade se todos os dados que contém são válidos, isto é, não contradizem a realidade que estão a representar nem se contradizem entre si. A manutenção da integridade pressupõe proteger a base de dados de acessos menos válidos por parte de utilizadores autorizados, impedindo-os de executar operações que ponham em risco a correção dos dados armazenados.

- *Capacidade de tolerância e recuperação de falhas* - Devido à potencial importância dos dados armazenados numa base de dados, é essencial a implementação de mecanismos de tolerância a falhas (hardware / software), que garantam a reposição da informação para um estado anterior válido

  Para tal são usados alguns mecanismos:

    - Implementação de cópias de segurança com estados válidos da base de dados (Backups)
    - Registos de atividade (Logging). Registo de todas as operações efetuadas sobre a base de dados a partir do último ponto de cópia

Numa arquitetura cliente-servidor,

1. Cliente estabelece a ligação com o servidor;
2. Autenticação pelo sistema de gestão de base de dados (não pela máquina);
3. Cliente envia pedido ao servidor em SQL (*Structured Query Language*);
4. Servidor recebe pedido, verifica privilégios de utilizador;

5. Servidor responde a cliente (acesso aceite/negado).

O desenvolvimento dos sistemas de bases de dados compreende as seguintes etapas: *information system*, *database planning*, *system definition and user view*, *requirements collection and analysis*, *centralized and view integration approaches*, *database design*, *conceptual database design*, *logical database design*, *physical database design*, *database management system selection*, *application design*, *transaction*, *prototyping*, *implementation*, *data conversion and loading*, *testing*, *operational maintenance*. Segue os seguintes critérios de produção: validade estrutural, simplicidade, expressabilidade, não-redundância, partilhabilidade, extensibilidade, integridade, representação diagramática.

Algumas noções sobre consistência e integridade:

Inconsistência de dados: - Existe inconsistência de dados quando os mesmos dados estão armazenados em diferentes locais, mas com valores diferentes.

Integridade dos dados: - A integridade dos dados define-se como o estado onde todos os dados na base de dados estão consistentes com o mundo real.

Restrições de Integridade: - São regras que definem que dados são válidos. As restrições de integridade asseguram que alterações feitas na base de dados por utilizadores autorizados não levam a perda da consistência dos dados. Portanto, as restrições de integridade protegem a base de dados de danos acidentais.

Estado consistente de uma BD: - Estado de uma base de dados onde todas as restrições de integridade são satisfeitas.

Pergunta: Transacções: explique o que se entende por consistência. A quem compete assegurar a consistência, ao utilizador ou ao SGBD?

Resposta:

- ***Consistência*** - Estado da base de dados no qual todas as restrições de integridade são satisfeitas. Para assegurar a consistência da base de dados, as transacções devem iniciar-se com a base de dados num estado de consistência conhecido. Uma transacção faz evoluir a base de dados de um estado consistente para um novo estado de consistência.

- ***Responsabilidade*** - Ao programador que codifica a interação, compete assegurar a consistência de cada transação isolada, pois é o programador que conhece as regras do mundo real. Ao SGBD compete controlar a interaçãoii entre transações concorrentes para prevenir que destruam a consistência da base de dados, Nesta tarefa i SGBD usa uma variedade de mecanismos designados por esquemas de controlo de concorrência.

Uma **aplicação de base de dados** é um programa que interage com a base de dados. Um **sistema de base de dados** é uma coleção de aplicações que interagem com o sistema de gestão de base de dados e a própria base de dados. **Ficheiros** são coleções de registos com dados ordenados logicamente. **Metadados/Dicionário de Dados/Catálogo do Sistema** correspondem à descrição dos

dados. **Abstração de Dados** implica poder alterar a definição interna de um objeto sem afetar os seus utilizadores, desde que a definição externa se mantenha. Uma **entidade** é um objeto distinto na organização representada na base de dados, um **atributo** é uma propriedade que descreve algum aspeto do objeto, uma **relação** é uma associação entre entidades. A **SQL** é uma DML que permite acesso controlado à base de dados através de sistemas de segurança, integridade, controlo de concorrência, recuperação de dados e catálogo.

Um sistema de gestão de base de dados é constituído por:

- ***Hardware***:
  - *Backend* - parte do sistema de gestão de base de dados que acede à base de dados;
  - *Frontend* - parte do sistema de base de dados que faz interface com o utilizador;
- ***Software***;
- **Dados** - ponte entre os componentes tecnológicos e humanos;
- **Procedimentos** - instruções e regras que definem o desenho e o uso da base de dados;
- **Pessoas**:
  - Administrador de Dados - responsável pela gestão do recurso de dados, incluindo planeamento da base de dados, desenvolvimento e manutenção de padrões, políticas e procedimentos;
  - Administrador de Base de Dados - resposável pelo desenvolvimento da base de dados com desempenho satisfatório;
  - *Designers* de Base de Dados:
    - Lógicos (o quê) - identificam os dados e as relações/restrições entre eles;
    - Físicos (como) - decidem como implementar a base de dados;
  - Criadores de Aplicações - desenvolvem programas que possam ser usados pelos utilizadores para aceder à base de dados;
  - Utilizadores - ingénuos (se desconhecem o sistema de gestão de base de dados) ou sofisticados (se conhecem o sistema de gestão de base de dados).

Um sistema de gestão de base de dados permite: controlo de dados redundantes, consistência de dados, maior informação num menor conjunto de dados, partilha de dados, maior integridade de dados, maior segurança, aplicação de padrões, economia de escala, equilíbrio de requisitos conflituosos, fácil acesso e responsividade, maior produtividade, manutenção melhorada pela independência dos dados (alterações nas aplicações não afetam os dados), maior concorrência, melhores serviços de *back-up* e recuperação. No entanto, é mais complexo, ocupa mais espaço, tem custos maiores (*hardware*, *software* e conversão de dados), desempenho mutável, maior impacto em caso de falha.~

Capitulo II - Diagrama Entidade Associação

Modelos de dados são ferramentas de comunicação. Veem dados ao invés de aplicações, eliminam redundâncias, favorecem a partilha de dados entre aplicações. São constituídos pela identificação,

análise e registo de políticas da organização acerca dos dados e definidos a três níveis:

- *Concetual* - representação fiel da realidade, sem atender a quaisquer constrangimentos;
- *Lógico* - adaptação do modelo concetual a um modelo de dados específico independente de qualquer sistema de gestão de bases de dados;
- *Físico* - adaptação do modelo lógico às características do sistema informático.

As **técnicas de modelação** seguem: do particular para o geral (*bottom-up*, abordagem da Teoria da Normalização (Codd, 1970), adequada a pequenos projetos); do geral para o particular (*top-down*, abordagem do Modelo Entidade-Associação, adequada a grandes projetos).

O **modelo entidade-associação** é especificado a nível gráfico (através de um diagrama entidade-associação) e a nível descritivo (pela especificação de cada componente do modelo). Uma *entidade* é qualquer objeto ou conceito com interesse para a organização a respeito de qual é guardada informação, representada por substantivos inequívocos. Um *atributo* é qualquer propriedade de uma entidade, indivisível, representada por verbos, que pode ser identificador (chaves) ou descritor. Uma *associação* relaciona duas entidades entre si (associação binária), várias entidades entre si (associação complexa) ou uma entidade consigo própria (associação unária). Um **diagrama entidade-associação** é um grafo que representa entidades, associações e associações com atributos, com notação de Chen (1976) ou notação pé-de-galinha.

Uma associação pode ter diferentes graus:

- **1:1** - a cada ocorrência da entidade A está associada apenas uma ocorrência da entidade B (ou nenhuma); a cada ocorrência da entidade B está associada apenas uma ocorrência da entidade A (ou nenhuma);
- **1:N** - a cada ocorrência da entidade A está associada uma, várias ou nenhuma ocorrência da entidade B; a cada ocorrência da entidade B está associada apenas uma ocorrência da entidade A (ou nenhuma);
- **N:M** - a cada ocorrência da entidade A está associada uma, várias ou nenhuma ocorrência da entidade B; a cada ocorrência da entidade B está associada uma, várias ou nenhuma ocorrência da entidade A. Estas associações não são suportadas pelo sistema de gestão de base de dados, pelo que devem ser decompostas em associações 1:N.

Uma entidade pode participar de forma **obrigatória** (todas as ocorrências dessa entidade têm de estar associadas a alguma ocorrência de outra entidade que participe na associação) ou de forma **não-obrigatória** (a entidade pode ter ocorrências não associadas a qualquer ocorrência de outra entidade que participe na organização).

Associações complexas relacionam mais do que duas entidades entre si e devem ser usadas apenas quando o conceito inerente à associação não pode ser representado por um conjunto de associações binárias. Pode ter diferentes graus:

- **1:1:1** - a cada par de ocorrências das entidades B e C está associada apenas uma ocorrência da entidade A (ou nenhuma). Análogo para B e C;

- **1:1:N** - a cada par de ocorrências das entidades B e C está associada uma, várias ou nenhuma ocorrência da entidade A; a cada par de ocorrências A e B/C está associada apenas uma ocorrência de C/B (ou nenhuma). Análogo para **1:N:M** e **N:M:P**.

A decomposição de associações M:N em 1:N permite: ressaltar a existência de entidades não identificadas no início; facilitar a análise do diagrama em termos de consistência; dar ao modelo uma forma adequada para passos subsequentes do método. Não há transitividade, logo, só se aceitam as leituras diretamente representadas no diagrama entidade-associação. A não compreensão/definição de uma associação levam a armadilhas de ligação: *chasm trap* (ausência de informação, quando um modelo sugere a existência de uma relação entre entidades, mas não existe uma associação entre ocorrências de certas entidades) e *fan trap* (ambiguidade de informação, onde o modelo apresenta uma relação entre entidades, mas a associação entre ocorrências de certas entidades é ambígua).

Os probnlemas normalmente ocorrem devido a uma interpretação incorreta do significado das associações. De entre as armadilhas de ligação, duas têm especial interesse: *Ausência de Informação* (Abismo: `Chasm trap`) e a *Ambiguidade de informação* (Laço: ``Fan trap`)

Em geral, para identificar as armadilhas de ligação, deve-se assegurar que o significado da associação é totalmente compreendido e claramente definido. Se não compreendermos as associações, podemos criar um modelo que não é uma verdadeira representação do mundo real.

Fan trap - Onde um modelo representa uma relação entre dois tipos de entidade, mas o caminho entre ambos é ambíguo. Uma *Fan trap* pode ocorrer quando duas ou mais associações *1:N* se ligam à mesma entidade.

Chasm trap - Onde um modelo sugere que existe uma relação entre tipos de entidades mas o caminho entre elas não existe para certo tipo de ocurrencias.

A simplicidade e capacidade de representação do modelo entidade-associação contribuíram de forma significativa para o sucesso do modelo relacional. As associações são identificadas por concatenação dos identificadores das entidades associadas, exceto nos casos em que as entidades se associam várias vezes entre si. No geral, evitar ocorrências em que os identificadores de outras entidades tenham valores nulos, evitar repetição de identificadores, criar tabelas para associações apenas quando estritamente necessário. O modelo entidade-associação tem uma abordagem *top-down* pois começa por identificar os dados importantes (entidades e associações) a serem representados antes de adicionar detalhes, como informação sobre os dados (atributos) e restrições sobre estes. Uma **rede semântica** é um modelo orientado a objetos que representa entidades e associações com símbolos específicos.

Um **domínio de atributo** é o conjunto de valores possíveis para cada atributo; vários atributos podem partilhar um domínio e domínios podem conter subdomínios. Atributos podem ser: *simples/atómicos* (um único componente independente, que não pode ser dividido); *compostos* (vários componentes independentes que podem ser divididos); *de valor único* (possuem um valor único para cada ocorrência); *de valor múltiplo* (possuem vários valores para cada ocorrência); *derivados* (representam um valor derivável de outro atributo ou conjunto de atributos, não necessariamente do mesmo tipo). Atributos podem ser **chaves**, que podem ser: *candidatas* (conjunto mínimo de atributos que identifica

de modo único cada ocorrência de entidades, não suportando o valor nulo); *primárias* (identifica cada ocorrência de entidades de modo único); *compostas* (contém dois ou mais atributos).

Uma entidade pode ser **forte** (não depende da existência de outra, cada ocorrência é facilmente identificável) ou **fraca** (depende da existência de outra e não é possível identificá-la apenas com os seus atributos).

Capitulo III - Modelo de Dados

Um **modelo de dados** é uma coleção integrada de conceitos para descrever e manipular dados, relações e restrições entre dados, constituída por três componentes:

- **Parte Estrutural** - conjunto de regras a partir das quais uma base de dados pode ser construída;
- **Parte Manipulável** - tipo de operações que são permitidas sobre os dados (atualização e recuperação de dados, alteração estrutural da base de dados);
- **Conjunto de Restrições de Integridade** - assegura a precisão dos dados.

Os **modelos de dados baseados em objetos** usam conceitos como **entidade** (objeto distinto a representar numa base de dados), **atributo** (propriedade que descreve aspetos do objeto que se pretende guardar) e **relação** (associação entre entidades): entidade-relação, semântica, funcional, orientado a objetos. Os **modelos de dados baseados em registos** usam um número de registos de formato fixo, possivelmente de tipos diferentes, que definem um número fixo de campos, e são usados para especificar a estrutura geral da base de dados e uma descrição de alto nível da implementação, com o inconveniente de não providenciar instalações adequadas para especificar restrições sobre os dados:

- *Modelo de dados relacional* - baseado no conceito de relações matemáticas, em que dados e relações são representados como tabelas independentes entre si;
- *Modelo de dados de rede* - os dados são representados como coleções de registos e as relações como conjuntos, organizados em estruturas gráficas gerais como nós/segmentos (registos) e arestas (relações);
- *Modelo de dados hierárquico* - tipo restrito de modelo de dados de rede, em que os registos e as relações são representados num gráfico em árvore.

Pergunta, definição de relação: *Resp*: Dada uma coleção de conjuntos *D1*, *D2*, ... *Dn* (não necessariamente disjuntos), R é uma relação naqueles conjuntos se for constituída por um conjunto de n-uplos ordenados <*d1, d2, ... , dn*> tais que d1 pertence a D1, d2 pertence a D2, ..., dn pertence a Dn.

Para realizar interrogações acerca das propriedades das entidades representadas num modelo precisa-se de uma linguagem apropriada: **álgebra relacional**. Uma interrogação (*query*) em álgebra relacional consiste numa coleção de operações sobre relações. Cada operador aceita um ou dois operandos (relações) e devolve como resultado uma relação. A álgebra relacional é uma linguagem teórica com operações que atuam em uma ou mais relações para definir outra relação sem alterar as

relações originais. Como os operandos e os resultados são relações, a saída de uma operação pode tornar-se a entrada de outra, permitindo que as expressões sejam aninhadas - fecho. Cada *query* descreve um procedimento passo a passo para calcular a resposta desejada, baseado na ordem em que operadores são aplicados:

- **Projeção** (`π_x (R)`) - se a relação `R` é representada como uma tabela, a operação de projeção de `R` sobre o conjunto de atributos `x` é interpretada como a seleção das colunas de `R` que correspondem aos atributos de `x` e a eliminação das linhas duplicadas na tabela obtida;
- **Restrição/Seleção** (`σ_p (R)`) - sendo `R` interpretada como uma tabela, a operação de restrição poe ser interpretada como a eliminação das linhas da tabela `R` que não satisfazem a condição `p` ;
- **Operações com Conjuntos** - sejam `R1` e `R2` com igual número de atributos e mesmos domínios de atributos correspondentes (esquemas relacionais compatíveis):
  - **União** - `R1 ∪ R2` é o conjunto dos tuplos de `R1` e `R2` ;
  - **Diferença** - `R1 - R2` é o conjunto dos tuplos de `R1` que não pertencem a `R2` ;
  - **Interseção** - `R1 ∩ R2` é o conjunto dos tuplos comuns a `R1` e a `R2` ;
  - **Produto Cartesiano** - `R1 x R2` é a concatenação dos atributos de `R1` e `R2` ;
  - **Junção Natural** - relação cujo conjunto de atributos são os atributos de `A` e os atributos de `B` que não aparecem na condição de junção. Os tuplos da tabela são obtidos pela concatenação dos tuplos de `A` com os tuplos de `B` sempre que os valores de `x` sejam iguais;
  - **Equijunção** - relação cujo conjunto de atributos é formado pelos atributos de `A` e `B` . Os tuplos da tabela são obtidos pela concatenação dos tuplos de `A` com os tuplos de `B` sempre que os valores dos atributos de `x` são iguais aos atributos de `y` ;
  - **Junção com Condições** - relação que contém os tuplos do produto cartesiano de `R` e `S` que satisfazem a condição `F` . Quando a condição `F` contém apenas igualdades, obtém-se a *equijunção*;
  - **Junção Externa** - a *junção externa à esquerda* de duas relações `R` e `S` , `R ⟕ S` , é uma junção onde os tuplos de `R` que não tenham correspondência em `S` também são incluídos na relação resultado e os valores em falta na segunda relação são colocados a `null` . Analogamente para *junção externa à direita* (`R ⟖ S`) e *junção externa à esquerda e à direita* (`R ⟗ S`);
- **Divisão** - valores de `x` tais que o par `(x, y)` ocorre em `A` para todos os valores de `y` que ocorrem em `B` ;
- **Renomeação/Rebatização** - usar o operador `ρ` para explicitamente renomear relações ou atribuir um nome ao resultado de uma operação;
- **Operações de Agregação** - não são operações de álgebra relacional, mas são usados pelo operador de agrupamento. Aplicam-se aos atributos (colunas) e são usados para sumarizar ou agregar os valores de um atributo da relação (`COUNT` , `MAX` , `MIN` , `AVG` , `SUM`);
- **Operação de Agrupamento** - o operador `γ` permite formar grupos numa relação e/ou agregar colunas. Tem um índice, uma lista `L` , em que cada elemento é um atributo da relação `R` ao qual

é aplicado γ ou um operador de relação aplicado a um atributo da relação.

No **modelo relacional**, toda a informação é logicamente estruturada em relações (tabelas). Os objetivos do modelo relacional são: permitir um elevado grau de independência de dados; providenciar bases substanciais para lidar com problemas de semântica, consistência e redundância de dados, introduzindo o conceito de *relações normalizadas* (relações sem grupos repetidos); permitir a expansão de linguagens de manipulação de dados orientadas a conjuntos. Terminologia:

- *Relação* - tabela com colunas e linhas, usada para armazenar informação sobre os objetos a serem representados numa base de dados;
- *Atributo* - coluna nomeada de uma relação, cujas linhas correspondem a registos individuais, podendo surgir em qualquer ordem sem alterar a relação;
- *Domínio* - conjunto de valores permitidos para um ou mais atributos, permitindo ao utilizador definir num local central o significado e origem dos valores que um atributo pode possuir;
- *Tuplo* - linha de uma relação, podendo aparecer em qualquer ordem sem alterar a relação. A estrutura de uma relação, em conjunto com a especificação dos domínios e restrições de valores possíveis, usualmente fixa, é conhecida como *intenção*. Os tuplos são alteráveis ao longo do tempo e conhecidos como *extensão* ou *estado* de uma relação;
- *Grau* - número de atributos de uma relação (um, relação unária; dois, relação binária; três, relação ternária; quatro ou mais, relação n-ária);
- *Cardinalidade* - número de tuplos de uma relação, que altera de cada vez que se acrescentam/eliminam tuplos;
- *Base de Dados Relacional* - coleção de relações normalizadas com nomes distintos.

O **produto cartesiano** de dois conjuntos é o conjunto de todos os pares tais que o primeiro elemento é membro do primeiro conjunto e o segundo elemento é membro do segundo conjunto. Qualquer subproduto de um produto cartesiano é uma relação. Um **esquema de relação** é uma relação nomeada constituída por pares de conjuntos de atributos e nomes de domínios. Um **esquema de base de dados relacional** é um conjunto de esquemas de relação, cada um com nomes distintos. As relações integram as seguintes propriedades:

- Cada relação num esquema relacional tem um nome único;
- Cada célula de uma relação tem um valor único (atómico);
- Cada atributo tem um nome distinto;
- Todos os valores de um atributo pertencem ao mesmo domínio;
- Cada tuplo é único;
- A ordem dos atributos não tem significado;
- A ordem dos tuplos não tem significado, teoricamente.

Chaves relacionais identificam um ou mais atributos que identificam de modo único cada tuplo numa relação:

- *Superchave* - atributo ou conjunto de atributos que identifica de modo único um tuplo interno a uma relação;
- *Chave Candidata* - superchave de forma que nenhum subconjunto próprio seja uma superchave dentro da relação, devendo ser única e irredutível. Quando a chave consiste em mais que um atributo, dá-se o nome de *chave composta*;
- *Chave Primária* - chave candidata que é selecionada para identificar tuplos exclusivamente dentro da relação. As outras chaves candidatas são chamadas de *chaves alternativas*;
- *Chave Estrangeira* - atributo ou conjunto de atributos internos a uma relação que corresponde(m) à chave candidata de outra relação.

Representação de esquemas de bases de dados relacionais: dar o nome de uma relação seguida dos respetivos atributos entre parêntesis, com a chave primária sublinhada.

Restrições de Integridade - como cada atributo tem um domínio associado, existem restrições (restrições de domínio) que formam restrições ao conjunto de valores permitidos para os atributos de relações:

- *Null* - representa um valor para um atributo que é desconhecido no momento ou não aplicável a um tuplo;
- *Integridade de Entidade* - numa relação de base, nenhum atributo de uma chave primária pode ser *null*;
- *Integridade Referencial* - se numa relação existe uma chave estrangeira, ou o valor da chave estrangeira corresponde a um valor da chave candidata de algum tuplo da relação inicial ou o valor da chave estrangeira é *null* na sua totalidade;
- *Restrições Gerais* - regras adicionais especificadas por utilizadores ou administradores da base de dados que definem ou restringem algum aspeto da organização.

Regras de Codd

Um sistema de gestão de base de dados relacional mantém informação de todas as tabelas e índices que contém. O **catálogo** tem dados globais ao sistema, estatísticas sobre tabelas e índices (cardinalidade, tamanho, altura, gama), informação sobre utilizadores e privilégios e o próprio catálogo. Segue as doze regras de Codd:

1. Todos os dados, incluindo o próprio dicionário de dados, são representados de uma só forma;
2. Cada elemento é bem determinado pela combinação do nome da tabela onde está armazenado, valor da chave primária e respetiva coluna (atributo);
3. Valores nulos (`NULL`) são suportados para representar informação não disponível ou não aplicável;
4. Os metadados são representados e acedidos da mesma forma que os próprios dados;
5. Deve existir pelo menos uma linguagem que: manipule dados; defina dados, *views*, restrições de integridade, acessos/autorizações; manipule transações;

6. Numa *view*, todos os dados atualizáveis que forem modificados devem ver essas modificações traduzidas nas tabelas-base;
7. Capacidade de tratar uma tabela como se fosse um simples operando;
8. Independência física - alterações na organização física dos ficheiros da base de dados ou nos métodos de acesso a esses ficheiros (nível interno) não devem afetar o nível concetual;
9. Independência lógica - alterações no esquema da base de dados (nível concetual), que não envolvam remoção de elementos, não devem afetar o nível externo;
10. As restrições de integridade devem poder ser especificadas numa linguagem relacional e armazenadas no dicionário de dados;
11. O facto de uma base de dados estar centralizada ou dispersa não deve repercutir-se na manipulação dos dados;
12. Se existir no sistema uma linguagem de mais baixo-nível, esta não pode permitir ultrapassar as restrições de integridade e segurança.

A SQL é uma linguagem não procedimental, ou seja, especifica-se a informação desejada, mas não o modo de obtenção desta. Um *query block* é um bloco-base de interrogação que permite a implementação das operações de seleção, projeção e junção da álgebra relacional, mas não especifica a ordem pela qual as operações são executadas. Linguagens de quarta geração são geradores de formulários, relatórios, gráficos e aplicações.

Capitulo IV - Base de Dados

Teoria da Normalização - obtenção de um modelo que garanta redundância mínima, facilidade de manutenção e estabilidade face a futuras alterações. É um processo de estruturação de tabelas de uma base de dados de modo a minimizar a redundância dos dados nela armazenados.

Dados redundantes - podem existir dados repetidos úteis (duplicados) em que, removendo-os, há perda de informação. Dados repetidos redundantes são dados que, sendo apagados de determinado tuplo, podem ser obtidos a partir de outro. Relações que possuem dados redundantes podem vir a ter *anomalias* de:

- *Inserção* - tem de se inserir toda a informação duplicada, comcuidado para não criar inconsistências com a informação já existente; não é possível criar novos tuplos com informação necessária se a chave da relação for nula;
- *Eliminação* - um tuplo que possua informação única, se for removido, leva à perda desses mesmos dados;
- *Modificação* - para alterar um atributo de um dado tuplo, devem atualizar-se todos os outros tuplos que possuam o mesmo atributo.

Dependências Lógicas - associações lógicas descritas entre os atributos de uma relação. Podem ser funcionais, multivalor ou de junção.

Dependências Funcionais - seja `R(A_1, A_2, ..., A_n)` um esquema de relação e `X` e `Y` subconjuntos de `{A_1, A_2, ..., A_n}` não vazios. Há uma dependência funcional entre `X` e `Y` (`X → Y` (X determina Y)) se em qualquer instante `t` quaisquer tuplos de `R` com o mesmo valor de `X` têm necessariamente o mesmo valor de `Y`.

Chave Candidata - seja a relação `R(A_1, A_2, ..., A_n)` e `x` um conjunto de atributos de `R` (`x ⊆ {A_1, A_2, ..., A_n}`). `x` é chave de `R` se:

- `∀i x → A_i, i = 1, 2, ..., n` - determina funcionalmente todos os atributos de `R`;
- `∄(y ⊊ x): ∀y → A_i, i = 1, 2, ..., n` - não existe um subconjunto próprio de `x` que determine todos os atributos de `R`.

Superchave - qualquer conjunto de atributos `x` que determine funcionalmente todos os atributos de `R`.

Chave Primária - chave candidata escolhida em que nenhuma das ocorrências pode ter valor nulo.

Toda a relação tem uma chave.

Propriedades básicas das dependências funcionais - **axiomas de Armstrong**:

- *Unicidade* - se `f: X → Y` e `g: X → Y`, então `f = g`;
- *Reflexibilidade* - se `X ⊇ Y`, então `X → Y`;
- *Transitividade* - se `X → Y` e `Y → Z`, então `X → Z`;
- *Aumento* - se `X → Y`, então `XZ → YZ`, para qualquer `Z`.

Propriedades derivadas das dependências funcionais:

- *Distributividade* (decomposição) - se `X → YZ`, então `X → Y` e `X → Z`;
- *União* - se `X → Y` e `X → Z`, então `X → YZ`;
- *Pseudotransitividade* - se `X → Y` e `YW → Z`, então `XW → Z`.

Dependência Funcional Trivial - todo o conjunto de atributos determina todos os seus subconjuntos. Se `X` determina `Y` (`Y` depende de `X`) então também `Y` determina `X` e `X` depende de `Y`. Se o atributo dependente não pertence ao determinadte, a dependência é não trivial.

Diz-se que uma dependência funcional `f` é **implicada** por um dado conjunto `F` de dependências funcionais se `f` é válida em cada instância da relação que satisfaz as dependências de `F`, ou seja, `f` é válida sempre que todas as dependências funcionais de `F` se verificam. O conjunto das dependências funcionais implicadas por um dado conjunto `F` de dependências funcionais chama-se **fecho de F**, `F+`. Este conjunto de dependências funcionais aplicadas é obtido pela aplicação repetitiva dos axiomas de Armstrong.

Os axiomas de Armstrong são **sólidos**, pois só geram dependências funcionais válidas (pertencentes a `F+`) quando aplicados a um conjunto `F` de dependências funcionais. São também **completos**, pois a aplicação repetida das regras gera todas as dependências funcionais de `F+`.

O **fecho de um conjunto de atributos** `X+`, tendo em conta o conjunto `F` de dependências funcionais, é o conjunto de atributos `A` tais que `X → A` pode ser inferido aplicando os axiomas de Armstrong.

Sejam `S1` e `S2` dois conjuntos de dependências funcionais. Se qualquer dependência funcional implicada por `S1` também é implicada por `S2` (`S1+` é um subconjunto de `S2+`) diz-se que `S2` é uma **cobertura** de `S1`. Se `S2` é uma cobertura de `S1` e `S1` é uma cobertura de `S2` (`S1+ = S2+`), diz-se que `S1` e `S2` são **equivalentes**.

Um conjunto `S` de dependências funcionais diz-se **irredutível** se e só se: o lado dependente de qualquer dependência funcional em `S` tem apenas um atributo; o lado determinante não pode ser alterado sem alterar `S+`; nenhuma das dependências funcionais de `S` pode ser removida de `S` sem alterar o fecho de `S`.

O conjunto `{A_1, A_2, ..., A_n}+` é o conjunto de todos os atributos de uma relação se e só se `A_1, A_2, ..., A_n` for uma superchave da relação.

Decomposição Sem Perdas - seja a relação `R` com `X`, `Y`, `Z` atributos. Se `X → Y` ou `X → Z`, então `R(X, Y, Z) = π X,Y (R) ⋈ π X, Z (R)`

Normalização - estudo das relações (dependências funcionais) entre atributos de modo a identificar o agrupamento ótimo para estes atributos para identificar um conjunto de relações adequado para satisfazer os requisitos da base de dados. Este grupo deve: possuir o mínimo de atributos possível; possuir na mesma relação atributos com uma relação lógica próxima (dependência funcional); possuir mínima redundância, com cada atributo a ser representado uma única vez.

Agrupar atributos em relações para minimizar redundância de dados permite atualizações com um número mínimo de operações e redução de inconsistência de dados e redução do espaço de armazenamento necessário e utilizado e de custos. Relações com dados redundantes podem vir a ter **anomalias de atualização** (inserção, eliminação, modificação). Duas propriedades importantes relacionadas com decomposição de uma relação maior noutras mais pequenas: *junção sem perdas* (qualquer instância da relação original pode ser identificada pelas instâncias correspondentes nas relações menores) e *preservação de dependência* (uma restrição na relação original pode ser mantida ao reforçar restrições em cada relação menor).

Dependência Funcional - `B` é funcionalmente dependente de `A` (`A ® B`) se cada valor de `A` é associado a apenas um valor de `B`. A dependência é especificada como uma **restrição** entre os atributos. Uma dependência funcional é propriedade de um esquema relacional, não de uma instância particular do esquema.

Determinante - atributo ou grupo de atributos no lado esquerdo da seta de uma dependência funcional.

Uma **dependência funcional completa** implica que `B` é total e funcionalmente dependente de `A` se `B` é funcionalmente dependente de `A`, mas não dos seus subconjuntos (a eliminação de qualquer atributo de `A` implica a perda da dependência). Uma **dependência parcial** existe quando há atributos

de `A` que podem ser removidos sem quebrar a relação. Uma dependência funcional é permanente; tem uma relação de 1:1 entre os atributos determinantes e determinados; tem o determinante com o mínimo número de atributos necessário para manter a dependência (funcional total).

Dependência Transitiva - se `A ® B` e `B ® C`, então `C` é transitivamente dependente de `A` por `B`.

Identificar o conjunto de dependências funcionais especifica o conjunto de restrições de integridade que devem existir numa relação.

A normalização de uma relação é obtida pela sua decomposição em duas ou mais relações. Uma relação numa forma normal mais avançada tem menos dados redundantes; se uma relação está numa forma normal mais avançada, também está nas formas normais anteriores:

- **Primeira Forma Normal (1FN)** - uma relação `R(A_1, A_2, ..., A_n)` é designada por *relação universal* se contiver todos os atributos relevantes da organização, cada um com um nome distinto. Uma relação está na primeira forma normal se: cada atributo contém apenas valores atómicos; não há conjuntos de atributos repetidos descrevendo a mesma característica;
- **Segunda Forma Normal (2FN)** - seja `R(X, Y, Z)` com `X`, `Y`, `Z` conjuntos de atributos, a dependência funcional `X → Y` é *elementar* se `∀ X' ⊂ X : X' ↛ Y`. Se `X → Y` e `X' → Y` (com `X' ⊂ X`), diz-se que `X → Y` é uma *dependência funcional parcial*. Seja `R(A_1, A_2, ..., A_n)`, `R` está na segunda forma normal se `∀ A_i ∉ chaves`, `∀ x chave`, se verifica que `X → A_i` é elementar, ou seja, se está na primeira forma normal e os atributos que não são chave dependem totalmente da chave. Um atributo pertencente a uma chave diz-se um *atributo primo*; casos especiais se todos os atributos de uma relação são primos; se a chave da relação consiste num único atributo. Para verificar se uma relação está na segunda forma normal, identifica-se a chave da tabela: se a chave for apenas um atributo ou for constituída por todos os atributos da relação, pode concluir-se que está na segunda forma normal; se a chave for composta, verifica-se se há atributos que não são chave e dependem apenas de parte da chave, se não houver, então está na segunda forma normal;
- **Terceira Forma Normal (3FN)** - seja `R(X, Y, Z)`, `X → Z` é *direta* se e só se `∄ Y ∈ R: Y ↛ X, X → Y, Y → Z` (não trivial). Seja `R(A_1, A_2, ..., A_n)`, `R` está na terceira forma normal se `∀ A_i ∉ chaves`, `∀ x chave`, se verifica que `X → A_i` é direta, ou seja, se está na segunda forma normal e nenhum dos atributos não chave depende de outro também não chave;
- **Forma Normal de Boyce-Codd** - seja `R(A_1, A_2, ..., A_n)`, `R` está na forma normal de Boyce-Codd se, para qualquer dependência funcional `X → Y` (não trivial), `X` e `Y` conjuntos de atributos de `R`, `X` é chave candidaaa de `R`, ou seja, se todo o determinante da relação for uma chave candidata. A forma normal de Boyce-Codd só é diferente da terceira forma normal se a relação tem mais do que uma chave. Se uma relação está na forma normal de Boyce-Codd, está na terceira forma normal.

As dependências funcionais representam restrições de integridade e, portanto, devem ser mantidas nas relações resultantes da decomposição. Seja `R` um esquema de relação que é decomposto em dois esquemas de relação com conjuntos de atributos `X` e `Y` e seja `F` um conjunto de dependências funcionais em `R`. A **projeção de `F` sobre `X` é o conjunto de dependências

funcionais no fecho `F+` que contém apenas atributos em `X - Fx`. Uma dependência funcional `U → V` em `F+` só pertence a `Fx` se todos os atributos em `U` e `V` pertencem a `X`. A decomposição da relação `R` com dependências funcionais `F` em esquemas de relação com conjuntos `X` e `Y` preserva as dependências se `(Fx U Fy)+ = F+`, ou seja, se se tomarem as dependências em `Fx` e `Fy` e se calcular o fecho da sua união se obtêm todas as dependências do fecho de `F`. Uma aplicação direta da definição dá um algoritmo direto para testar quando é que uma decomposição preserva as dependências funcionais.