Sem título

BD resumos

a. Comparação entre Sistemas OLTP e Data Warehousing
    i. Problemas Associados à Data Warehousing
b. 31.2 Data Warehouse Architecture
c. Data MArt
    i. 16.1.3 Fatores Críticos para o Sucesso no Design de Banco de Dados
    ii. 16.2 Visão Geral da Metodologia de Design de Banco de Dados
        1. Design Conceitual de Banco de Dados
        2. Design Lógico de Banco de Dados para o Modelo Relacional
        3. Design Físico de Banco de Dados para Bancos de Dados Relacionais

Comparação entre SGBD e sistemas baseados em ficheiros

Os Sistemas de Gerenciamento de Banco de Dados (SGBDs) oferecem uma solução mais robusta, flexível e escalável para o gerenciamento de dados em comparação com os sistemas baseados em arquivos. Eles suportam uma organização avançada dos dados, acesso eficiente, integridade e segurança aprimoradas, além de permitir uma manutenção e evolução mais simplificadas. Por outro lado, os sistemas baseados em arquivos, embora ainda úteis em alguns contextos específicos, apresentam limitações significativas em termos de flexibilidade, eficiência de acesso e capacidade de lidar com a complexidade e o volume crescente de dados.
Essa comparação evidencia a superioridade dos SGBDs para a maioria das aplicações modernas, onde a gestão eficiente e segura de grandes volumes de dados é crucial para o sucesso organizacional.

Situações Preferíveis para a Abordagem de Sistemas de Ficheiros

Quando a quantidade de informação armazenada é baixa e destinada a servir apenas um departamento, a abordagem de Sistemas de Ficheiros é preferível. Nesses cenários, a simplicidade dos Sistemas de Ficheiros oferece várias vantagens:

1. **Baixa Complexidade e Tamanho de Dados**: Para aplicações com dados simples e de pequeno volume, Sistemas de Ficheiros são mais adequados, já que não requerem a complexidade de um SGBD.
2. **Custo Inferior**: Sistemas de Ficheiros possuem custos de implementação e manutenção significativamente menores do que SGBDs, sendo ideais para operações com orçamento limitado.

3. **Menor Impacto de Falhas**: Em caso de falha, a recuperação de um Sistema de Ficheiros é geralmente mais simples e menos custosa, com impacto limitado devido ao escopo restrito da aplicação.
4. **Simplicidade Operacional**: Sistemas de Ficheiros são mais fáceis de gerenciar e operar, evitando a complexidade e o overhead de um SGBD, o que é benéfico para aplicações de curto prazo ou menos críticas.

Portanto, em ambientes onde a quantidade de dados é pequena, a necessidade de integração é mínima, e a eficiência de custo e simplicidade são primordiais, os Sistemas de Ficheiros são a escolha mais indicada.

DBA Função

O gestor de base de dados (DBA) é fundamental em uma solução baseada em Sistema de Gestão de Bases de Dados (SGBD). Suas principais funções incluem a instalação, configuração e manutenção do sistema, além da otimização do desempenho e da gestão da estrutura da base de dados. O DBA é responsável pela segurança dos dados, implementando controles de acesso e medidas de proteção contra acessos não autorizados e perdas de dados. Ele também realiza backups regulares e desenvolve planos de recuperação de desastres para garantir a continuidade das operações em casos de falhas.

Além disso, o gestor de BD oferece suporte técnico, resolve problemas operacionais e assegura que a base de dados esteja em conformidade com as normas regulatórias. Ele também planeja a capacidade futura e coordena a integração e migração de dados entre sistemas, mantendo a documentação detalhada de todas as atividades e procedimentos relacionados ao SGBD.

SGBD Funções

Um Sistema de Gestão de Bases de Dados (SGBD) deve satisfazer as seguintes funções essenciais:

1. **Armazenamento e Recuperação de Dados**: Gerenciar a persistência e a recuperação eficiente de grandes volumes de dados.
2. **Gestão de Transações**: Assegurar que operações múltiplas sejam executadas de maneira consistente e confiável.
3. **Segurança**: Controlar o acesso aos dados e proteger contra acessos não autorizados e corrupção de dados.
4. **Integridade dos Dados**: Garantir a precisão e a consistência dos dados armazenados.
5. **Backup e Recuperação**: Realizar backups regulares e fornecer mecanismos para recuperação em caso de falhas.

6. **Interface de Consulta**: Permitir que usuários consultem e manipulem dados de forma eficiente e intuitiva.
7. **Administração de Dados**: Facilitar a definição, manutenção e documentação das estruturas de dados.
8. **Suporte a Multiusuários**: Permitir que múltiplos usuários acessem e modifiquem os dados simultaneamente sem conflitos.

Gerações de SGBD

A evolução dos sistemas de banco de dados pode ser resumida em três gerações principais:

1. **Primeira Geração**: DBMSs hierárquicos e em rede, com foco na organização de dados de forma estruturada, mas com limitações na flexibilidade de consultas e na integração de dados.
2. **Segunda Geração**: DBMSs relacionais, que introduziram a independência de dados e uma linguagem padrão para consultas, mas com limitações na modelagem de dados.
3. **Terceira Geração**: DBMSs orientados a objetos e relacionais-objetos, que buscaram melhorar a capacidade de modelagem e integração com tecnologias emergentes como a web e XML.

Conclusão

A história dos DBMSs reflete uma evolução contínua em resposta às crescentes demandas de gerenciamento e análise de dados. Desde as primeiras estruturas hierárquicas e de rede até os avançados sistemas relacionais e orientados a objetos, cada etapa representou um avanço significativo na capacidade de lidar com volumes crescentes de dados e a complexidade das aplicações de negócios modernas.

Componentes de SGBD

Inter-relações entre os Componentes

- **Hardware e Software**: O software do SGBD é executado no hardware e precisa de recursos adequados para funcionar eficientemente.
- **Software e Dados**: O software gerencia o acesso, a manipulação e o armazenamento de dados.
- **Dados e Usuários**: Usuários acessam e manipulam dados através de interfaces fornecidas pelo software.
- **Usuários e Procedimentos**: Usuários seguem procedimentos para garantir o uso correto e seguro do SGBD.
- **Procedimentos e Hardware/Software**: Procedimentos são aplicados para manter o hardware e o software em bom funcionamento, além de assegurar a integridade e a segurança dos dados.

Descrições

descrição dos cinco componentes do ambiente de um SGBD, de forma resumida e na ordem solicitada:

1. **Hardware**

O hardware é a infraestrutura física que suporta o SGBD, incluindo servidores, dispositivos de armazenamento e redes. Ele hospeda o software e armazena os dados, fornecendo a base para todas as operações do SGBD.

2. **Software**

O software é o sistema de gerenciamento de banco de dados que executa operações de inserção, consulta, atualização e exclusão de dados. Ele gerencia o acesso aos dados, assegura a integridade e oferece interfaces para os usuários interagirem com os dados.

3. **Dados**

Os dados são a informação armazenada e gerenciada pelo SGBD, incluindo dados estruturados e não estruturados. Eles são organizados e acessados pelo software e são armazenados fisicamente no hardware.

4. **Procedimentos**

Procedimentos são as regras e diretrizes para operar e manter o SGBD, como backup, recuperação e segurança. Eles asseguram que o hardware e o software funcionem corretamente e que os dados sejam protegidos e acessíveis de forma segura.

5. **Usuários**

Os usuários interagem com o SGBD para realizar operações com os dados. Eles incluem administradores que mantêm o sistema, desenvolvedores que criam aplicativos e usuários finais que consultam e manipulam os dados.

Esses componentes são interdependentes, colaborando para proporcionar um ambiente de banco de dados funcional e seguro, com o hardware suportando o software, que gerencia os dados de acordo com procedimentos seguidos pelos usuários.

Arquitetura ANSI SPARC SGBD

Motivação

- Todos os utilizadores devem ter acesso aos mesmos dados.
- A perspectiva do utilizador deve ser imune às alterações feitas noutras perspectivas (ou vistas ou views).
- Os utilizadores não precisam de saber dos detalhes de armazenamento físico da base de dados.
- O DBA deve ser capaz de alterar as estruturas de armazenamento da base de dados sem afetar as vistas dos utilizadores.
- A estrutura interna da base de dados não deve ser afectada por alterações físicas de armazenamento.

● O DBA deve ser capaz de alterar a estrutura conceptual da base de dados sem afetar os utilizadores.

Niveis

Essa arquitetura de três níveis ajuda a promover a independência dos dados, permitindo que alterações em um nível não afetem diretamente os outros, além de fornecer uma visão flexível e personalizada para diferentes usuários, uma visão conceitual integrada e um gerenciamento físico eficiente dos dados.

2.1.1 External Level

O **nível externo** representa a visão dos usuários sobre o banco de dados. Cada usuário ou grupo de usuários possui uma visão personalizada do "mundo real" relevante para suas necessidades específicas, sem conhecer os detalhes completos do banco de dados. Essa visão inclui apenas as entidades, atributos e relações de interesse para o usuário, permitindo que diferentes usuários vejam os dados de maneiras distintas.
Por exemplo, a data de nascimento pode ser visualizada de formas diferentes por diferentes usuários, e dados derivados, como a idade calculada a partir da data de nascimento, podem ser apresentados sem estarem armazenados diretamente no banco de dados.

2.1.2 Conceptual Level

O **nível conceitual** oferece uma visão global e integrada do banco de dados,descrevendo o que é armazenado e as relações entre os dados. Ele é independente de considerações de armazenamento físico e inclui a definição de todas as entidades, seus atributos, relações, restrições, além de informações sobre segurança e integridade dos dados. O nível conceitual garante que qualquer dado disponível a um usuário no nível externo esteja contido ou seja derivável do esquema conceitual, proporcionando uma estrutura lógica que não depende de detalhes físicos de armazenamento.

2.1.3 Internal Level

O **nível interno** trata da representação física do banco de dados no sistema de armazenamento. Ele descreve como os dados são armazenados fisicamente, incluindo alocação de espaço, descrição de registros, técnicas de compressão e criptografia. Esse nível está preocupado com a eficiência de armazenamento e acesso aos dados, além de otimizar o desempenho de tempo de execução. O nível interno utiliza métodos de acesso do sistema operacional para gerenciar a disposição dos dados e a construção de índices, mas detalhes específicos de armazenamento podem variar conforme o sistema de banco de dados e o sistema operacional.

2.1.4 Schemas, Mappings, and Instances

- **Esquema da BD**: Descrição global da base de dados.
- **Esquema externo**: Conjunto de subesquemas que descrevem as perspectivas dos usuários, cada um focado em diferentes aspectos ou representações dos dados.
- **Esquema conceitual**: Descrição completa de todos os dados e suas relações, incluindo restrições de integridade. É um único esquema que oferece uma visão lógica unificada.
- **Esquema interno**: Descrição da estrutura física dos dados, incluindo métodos de armazenamento e organização dos registros. Existe apenas um esquema interno.
- **Instância**: Conjunto atual de dados armazenados num momento específico. As instâncias mudam com a adição, modificação ou remoção de dados.

O SGBD gerencia o mapeamento entre esses esquemas, garantindo que as mudanças em um nível não afetem os outros.

2.1.5 Data Independence

A **independência de dados** é um objetivo fundamental da arquitetura de três níveis, permitindo que mudanças em níveis inferiores não afetem os níveis superiores. Existem dois tipos principais de independência:

Há dois tipos de independência de dados:

- **Independência lógica**: Permite alterações no esquema conceitual sem impactar os esquemas externos ou aplicativos.
- **Independência física**: Permite mudanças no esquema interno, como métodos de armazenamento, sem afetar o esquema conceitual ou externo.

Essas independências garantem que a base de dados possa ser adaptada e otimizada com impacto mínimo nos usuários e nas aplicações.

2.2 Database Languages

2.2.1 The Data Definition Language (DDL)

- **DDL (Data Definition Language)**: Permite ao DBA ou usuário descrever e nomear entidades, atributos e relacionamentos necessários para a aplicação, juntamente com quaisquer restrições de integridade e segurança associadas.
- **Esquema da BD**: É especificado por meio de definições expressas em DDL, resultando na criação ou modificação de um esquema. O resultado das declarações DDL é um conjunto de tabelas armazenadas em arquivos especiais chamados catálogo do sistema.
- **Metadados**: Descrevem objetos na base de dados e facilitam seu acesso e manipulação. São consultados pelo DBMS antes dos dados reais serem acessados.
- **Data Dictionary**: Termo usado para descrever o catálogo do sistema em DBMS.

2.2.2 The Data Manipulation Language (DML)

- **DML (Data Manipulation Language)**: Oferece um conjunto de operações para manipulação básica de dados na base de dados, incluindo inserção, modificação, recuperação e exclusão de dados.

- **Linguagem de Consulta**: Uma linguagem especializada de alto nível usada para solicitar a recuperação de dados na base de dados.
- **Procedural DML**: Especifica como os dados devem ser obtidos, lidando com registros individualmente e frequentemente integrados a linguagens de programação.
- **Nonprocedural DML**: Especifica apenas quais dados são necessários, deixando para o DBMS decidir como obter os dados, comumente usado em linguagens declarativas como SQL.

IMPORTANCIA

As sublinguagens DDL e DML são essenciais para a administração e operação eficiente de bancos de dados. A DDL permite definir e modificar a estrutura da base de dados, garantindo consistência e integridade dos dados. Enquanto isso, a DML capacita usuários finais a manipular dados através de operações como inserção, atualização, recuperação e exclusão, facilitando interações dinâmicas e análises complexas. Essas linguagens são fundamentais para suportar diferentes necessidades de usuários, garantindo a segurança e integridade dos dados armazenados.

System Catalog

O system catalog serve para armazenar metadados que descrevem objetos na base de dados, como tabelas, índices, visões e restrições. Ele facilita o acesso e manipulação desses objetos pelo DBMS, fornecendo informações essenciais para garantir a integridade e consistência dos dados.

Modelo de Dados:

O modelo de dados é um conjunto de conceitos usados para descrever um conjunto de dados, as operações para manipular esses dados e um conjunto de restrições de integridade. Ele se divide em três categorias principais: modelos de dados baseados em objetos, modelos de dados baseados em registros e modelos de dados físicos. Os modelos baseados em objetos e baseados em registros são usados para descrever dados nos níveis conceitual e externo, enquanto o modelo físico descreve dados no nível interno.

Modelagem Conceitual:

A modelagem conceitual envolve a criação de uma arquitetura detalhada para um banco de dados que é independente de detalhes de implementação, como o sistema gerenciador de banco de dados (SGBD), programas de aplicação, linguagens de programação ou outras considerações físicas. O design do esquema conceitual é crucial para o sucesso geral do sistema, exigindo tempo e esforço adequados para produzir o melhor projeto conceitual possível.

O modelo conceitual desempenha um papel crucial no desenvolvimento de sistemas de banco de dados ao oferecer uma representação abstrata e independente de detalhes de implementação. Ele define a estrutura lógica dos dados, incluindo entidades, atributos,

relacionamentos e restrições de integridade, permitindo aos projetistas focarem nos requisitos do usuário e na modelagem precisa dos dados. Essa abordagem facilita a comunicação entre os envolvidos no projeto e proporciona uma base clara para o desenvolvimento de aplicações de banco de dados robustas e escaláveis.

Funções de um SGBD Multiusuário

As funções de um SGBD multiusuário incluem armazenamento, recuperação e atualização de dados; um catálogo acessível ao usuário; suporte a transações; controle de concorrência e serviços de recuperação; serviços de autorização; suporte à comunicação de dados; serviços de integridade; serviços que promovem a independência de dados; e serviços utilitários. Essas funções são essenciais para gerenciar operações de banco de dados de forma eficiente e segura, atendendo às necessidades de múltiplos usuários.

Database Architectures and the Web chap3

Comparação de arquitaturas principais

A arquitetura cliente-servidor de dois níveis conecta diretamente o cliente ao servidor de banco de dados, enquanto a de três níveisadiciona uma camada intermediária de aplicação entre eles. Para a Web, a arquitetura de três níveis é preferível devido à sua escalabilidade, segurança aprimorada, flexibilidade na manutenção e capacidade de otimização de desempenho através de caches distribuídos. Isso a torna mais adequada para lidar com o grande volume de usuários e requisições típicos de aplicações web modernas.

Multi-user DBMS Architectures

Claro! Aqui está o resumo adicionando também a arquitetura de file-server:

1. **Teleprocessamento:**
- Arquitetura tradicional para sistemas multiusuário.
- Um único computador central (mainframe) com uma CPU onde todos os processamentos ocorrem.
- Terminais "burros" conectados via rede enviam e recebem mensagens através do sistema operacional.
- Carga significativa no computador central para executar programas de aplicação, DBMS e processamento para terminais.
- Desvantagens incluem sobrecarga no mainframe e complexidade na manutenção.

2. **Arquitetura Cliente-Servidor de Dois Níveis:**
- Evolução do teleprocessamento para descentralizar a computação.

- Dividida em cliente (responsável pela apresentação de dados ao usuário) e servidor (responsável pelos serviços de dados).
- Melhora o desempenho e a escalabilidade distribuindo parte do processamento para clientes.
- Ainda requer uma cópia completa do DBMS em cada estação de trabalho, o que pode ser redundante e complexo em termos de controle de concorrência e recuperação.

3. **Arquitetura File-Server:**
- Ambiente onde o processamento é distribuído, mas os dados são centralizados em um servidor de arquivos.
- Utilizado principalmente em redes locais (LANs), onde um servidor de arquivos contém os dados acessados pelas aplicações e pelo DBMS em cada estação de trabalho.
- As aplicações e o DBMS em cada estação de trabalho solicitam arquivos do servidor de arquivos conforme necessário.
- Gera um tráfego significativo na rede, podendo resultar em problemas de desempenho.
- Requer uma cópia completa do DBMS em cada estação de trabalho, complicando o controle de concorrência, recuperação e integridade devido ao acesso concorrente aos mesmos arquivos.

4. **Arquitetura Cliente-Servidor de Três Níveis:**
- Desenvolvida para superar limitações do modelo de dois níveis, especialmente em termos de escalabilidade empresarial.
- Camadas distintas para interface do usuário (cliente), lógica de negócios e processamento de dados (aplicação servidor) e DBMS (servidor de banco de dados).
- Cliente magro que cuida apenas da interface do usuário, transferindo a maior parte da lógica de aplicação e processamento para o servidor de aplicação.
- Centraliza a manutenção da aplicação e simplifica a distribuição de software.
- Facilita a escalabilidade, o balanceamento de carga e é ideal para ambientes web, onde se adapta naturalmente ao navegador como cliente e servidor web como servidor de aplicação.

Essas arquiteturas refletem a evolução das necessidades empresariais e tecnológicas ao longo do tempo, cada uma trazendo vantagens distintas para diferentes cenários de implementação de sistemas multiusuário.

4. Relational model

Este modelo surgiu em 1970, com base na publicação:

- Codd, E. F., “A relational Model for Large Shared Data Banks”, Communications of the ACM, 13, 6 (1970), 377-387

Diferente do modelo anterior, que foi basicamente a evolução do modelo hierárquico, ambos baseados nas técnicas de processamento de arquivos, este modelo baseou-se no modelo matemático simples e ao mesmo tempo eficiente: teoria dos conjuntos.

4.2 terminologia

Relação:

- Uma relação é uma tabela com colunas e linhas.
- No contexto de um SGBD relacional, os dados são percebidos pelo usuário como tabelas, aplicando-se apenas à estrutura lógica do banco de dados.

Atributo:

- Um atributo é uma coluna nomeada de uma relação.
- Representa uma característica ou propriedade dos objetos na base de dados.

Domínio:

- Um domínio é o conjunto de valores permitidos para um ou mais atributos.
- Domínios ajudam a definir centralmente os valores que atributos podem ter, prevenindo operações semânticamente incorretas.

Tupla:

- Uma tupla é uma linha de uma relação.
- Cada linha na tabela representa um registro individual, e a ordem das tuplas não altera a relação.

Grau:

- O grau de uma relação é o número de atributos que ela possui.
- Exemplo: uma relação com quatro atributos tem grau quatro.

Cardinalidade:

- A cardinalidade de uma relação é o número de tuplas que ela contém.
- Reflete a quantidade de registros em um dado momento.

Banco de Dados Relacional:

- É uma coleção de relações normalizadas com nomes distintos.
- A normalização assegura que as relações estejam estruturadas adequadamente para evitar redundâncias e garantir a integridade dos dados.

4.2.4 Propriedades das Relações no Modelo Relacional

1. **Nome Distinto:**
   - Cada relação tem um nome único no esquema relacional, distinguindo-a de outras relações.
2. **Valores Atômicos:**

- Cada célula de uma relação contém exatamente um valor atômico, ou seja, um único valor indivisível.

3. **Atributos com Nomes Distintos:**
   - Cada atributo dentro de uma relação possui um nome único.
4. **Valores do Mesmo Domínio:**
   - Os valores de um atributo provêm do mesmo domínio, assegurando a consistência dos dados.
5. **Tuplas Únicas:**
   - Cada tupla (linha) é distinta, não havendo duplicação de registros.
6. **Ordem dos Atributos Irrelevante:**
   - A ordem das colunas (atributos) não afeta a definição da relação.
7. **Ordem das Tuplas Irrelevante:**
   - Teoricamente, a ordem das tuplas (linhas) não importa para a relação. Na prática, a ordem pode afetar a eficiência do acesso aos dados.

Essas propriedades garantem que as relações no modelo relacional sejam bem definidas, consistentes e livres de duplicidade, além de facilitarem a organização e a manipulação eficiente dos dados.

4.2.5 Chaves relacionais

Superkey

- Um **superkey** é um ou um conjunto de atributos que identifica unicamente uma tupla em uma relação. Ele pode conter mais atributos do que o necessário para essa identificação.

Candidate Key

- Uma **candidate key** é um superkey minimal, ou seja, não possui nenhum subconjunto que também seja um superkey. Tem duas propriedades principais:
  - **Unicidade:** Os valores identificam unicamente uma tupla.
  - **Irredutibilidade:** Nenhum subconjunto pode identificar unicamente as tuplas.
- Pode haver várias candidate keys em uma relação, e quando uma chave é composta por mais de um atributo, é chamada de **chave composta**.

Primary Key

- A **primary key** é a candidate key selecionada para identificar unicamente as tuplas de uma relação.
- Cada relação deve ter uma primary key, e as candidate keys que não são escolhidas como primary key são chamadas de **chaves alternativas**.

Foreign Key

- Uma **foreign key** é um ou um conjunto de atributos em uma relação que corresponde à candidate key de outra (ou da mesma) relação.
- Serve para estabelecer relações entre diferentes tabelas, como associar funcionários a seus respectivos departamentos através de um identificador comum.

Essas chaves são fundamentais para manter a integridade e a coerência dos dados em um banco de dados relacional, permitindo a identificação única de registros e a criação de relacionamentos entre tabelas.

4.3 Restrições de integridade

As principais restrições de integridade são as seguintes:

1. Null

- **Definição:** Representa um valor para um atributo que é desconhecido ou não aplicável para uma tupla.
- **Função:** Indica ausência de valor, diferenciando-se de valores como zero ou espaços em branco, que são significativos. Nulls ajudam a lidar com dados incompletos ou excepcionais, mas podem causar problemas na implementação devido à sua incompatibilidade com a lógica booleana (verdadeiro/falso).

2. Integridade de Entidade

- **Definição:** Nenhum atributo de uma chave primária em uma relação base pode ser nulo.
- **Objetivo:** Garante que cada tupla possa ser identificada unicamente por sua chave primária. Permitir null em qualquer parte da chave primária implica que a chave completa não é necessária para a identificação única, o que contraria a definição de chave primária. Aplicável especificamente a relações base.

3. Integridade Referencial

- **Definição:** Se uma chave estrangeira existir em uma relação, seu valor deve corresponder a uma chave candidata de alguma tupla em sua relação de origem ou deve ser totalmente nulo.
- **Objetivo:** Mantém a consistência entre relações relacionadas. Por exemplo, um registro de funcionário não pode referenciar um número de filial inexistente, a menos que o número de filial seja nulo, permitindo que um funcionário não alocado a uma filial ainda seja registrado.

4. Restrições Gerais

- **Definição:** Restrições adicionais especificadas por usuários ou administradores de banco de dados que definem ou limitam algum aspecto da organização.
- **Objetivo:** Permite a imposição de regras específicas, como um limite no número de funcionários por filial. O suporte a essas restrições varia entre os sistemas de gerenciamento de banco de dados (SGBDs).

4.4 Views

Vistas vs. Relações Base

- **Relação Base:** São relações reais cujas tuplas são fisicamente armazenadas no banco de dados e correspondem a entidades no esquema conceitual. São as tabelas concretas onde os dados são mantidos.
- **Vista:** É uma relação virtual ou derivada, resultado dinâmico de operações sobre uma ou mais relações base. Não existe fisicamente como os dados das relações base, mas é definida por uma consulta que pode ser executada sob demanda. A definição da vista é armazenada no catálogo do sistema, mas os dados em si não são mantidos separadamente.

Objetivo/vantagens da Criação de Vistas

1. **Segurança:** Escondem dados confidenciais, permitindo acesso controlado.
2. **Personalização:** Apresentam os dados de forma customizada para diferentes usuários.
3. **Simplificação:** Facilitam operações complexas ao permitir que os usuários trabalhem com uma interface mais simples.
4. **Independência Lógica:** Isolam os usuários de mudanças na estrutura do banco de dados, mantendo a interface consistente.

Atualização de Vistas

- **Permitidas em Vistas Simples:** Atualizações são permitidas em vistas que se baseiam em uma única relação base contendo uma chave primária ou candidata.
- **Restrições:** Não é permitido atualizar vistas que envolvem múltiplas tabelas ou operações de agregação, pois essas mudanças não podem ser facilmente refletidas nas tabelas originais.

5. Álgebra relacional

Características da Álgebra Relacional

A álgebra relacional é uma linguagem teórica para manipulação de dados em bancos de dados relacionais. Suas principais características são:

- **Fechamento:** As operações são realizadas sobre relações e geram novas relações, permitindo o aninhamento de operações.
- **Linguagem Conjunta:** Manipula conjuntos de tuplas de uma só vez, sem necessidade de loops.
- **Procedural:** Define explicitamente o passo a passo para transformar dados, diferente de linguagens declarativas como SQL.

Operações Unárias

1. **Seleção (σ)**
   - **Descrição:** Filtra tuplas de uma relação que satisfazem um predicado especificado.
   - **Uso:** σpredicado(R)\sigma_{predicado}(R)σpredicado(R)
   - **Exemplo:** σidade>30(Funciona´rios)\sigma_{idade > 30}(\text{Funcionários})σidade>30(Funciona´rios) retorna os funcionários com idade superior a 30.
2. **Projeção (π)**
   - **Descrição:** Extrai um subconjunto vertical de atributos de uma relação, eliminando duplicatas.
   - **Uso:** πa1,...,an(R)\pi_{a1, ..., an}(R)πa1,...,an(R)
   - **Exemplo:** πnome,idade(Funciona´rios)\pi_{nome, idade}(\text{Funcionários})πnome,idade(Funciona´rios) retorna uma relação com os atributos nome e idade dos funcionários.

Operações de Conjunto (Binárias)

3. **União (∪)**
   - **Descrição:** Combina todas as tuplas de duas relações, eliminando duplicatas.
   - **Uso:** R∪SR \cup SR∪S
   - **Exemplo:** Funciona´rios∪Gerentes\text{Funcionários} \cup \text{Gerentes}Funciona´rios∪Gerentes retorna todas as tuplas presentes em Funcionários ou Gerentes.
4. **Diferença de Conjuntos (−)**
   - **Descrição:** Retorna tuplas presentes em uma relação que não estão na outra.
   - **Uso:** R−SR - SR−S
   - **Exemplo:** Funciona´rios−Gerentes\text{Funcionários} - \text{Gerentes}Funciona´rios−Gerentes retorna as tuplas de Funcionários que não estão em Gerentes.
5. **Interseção (∩)**
   - **Descrição:** Retorna tuplas presentes em ambas as relações.
   - **Uso:** R∩SR \cap SR∩S
   - **Exemplo:** Funciona´rios∩Gerentes\text{Funcionários} \cap \text{Gerentes}Funciona´rios∩Gerentes retorna as tuplas que estão tanto em Funcionários quanto em Gerentes.
6. **Produto Cartesiano (×)**
   - **Descrição:** Combina cada tupla de uma relação com cada tupla de outra relação.
   - **Uso:** R×SR \times SR×S
   - **Exemplo:** Funciona´rios×Departamentos\text{Funcionários} \times \text{Departamentos}Funciona´rios×Departamentos gera todas as combinações possíveis de Funcionários e Departamentos.

Operações de Junção

6. **Junção (Join)**

- **Theta Join (R ⋈F S):** Combina tuplas de R e S que satisfazem um predicado F.
- **Equijoin (R ⋈F S):** Combina tuplas de R e S usando apenas operações de igualdade.
- **Natural Join (R ⋈ S):** Equijoin de R e S sobre todos os atributos comuns, eliminando duplicatas.

7. **Outer Join (R ⟕ S)**
   - **Left Outer Join (R ⟕ S):** Inclui todas as tuplas de R e as tuplas de S que têm correspondência, incluindo tuplas de R sem correspondência em S.
8. **Semijoin (R ⋉F S)**
   - **Descrição:** Retorna as tuplas de R que participam da junção de R com S, satisfazendo o predicado F.

Operação de Divisão

10. **Divisão (R ÷ S)**
    - **Descrição:** Retorna tuplas de R que estão associadas a todas as tuplas de S.
    - **Exemplo:** R÷SR \div SR÷S retorna as tuplas de R que têm uma combinação para cada tupla em S.

Operações de Agrupamento

11. **Agregação (Aggregate AL(R))**
    - **Descrição:** Aplica a lista de funções de agregação AL à relação R para definir uma nova relação sobre a lista de agregados. AL contém um ou mais pares (func¸a˜o_de_agregac¸a˜o,atributo)(\text{função_de_agregação}, \text{atributo}) (func¸a˜o_de_agregac¸a˜o,atributo).
    - **Exemplo:** Aggregate{SUM(sala´rio)}(Funciona´rios)\text{Aggregate} \{ \text{SUM} (salário) \}(Funcionários)Aggregate{SUM(sala´rio)}(Funciona´rios) retorna a soma de salários de todos os funcionários.
12. **Agrupamento (Grouping GA AL(R))**
    - **Descrição:** Agrupa as tuplas da relação R pelos atributos de agrupamento GA e, em seguida, aplica a lista de funções de agregação AL para definir uma nova relação. A relação resultante contém os atributos de agrupamento GA junto com os resultados de cada função de agregação.
    - **Exemplo:** Grouping{depto_id}{AVG(sala´rio)}(Funciona´rios)\text{Grouping} \{ \text{depto_id} \} \{ \text{AVG}(salário) \}(Funcionários)Grouping{depto_id} {AVG(sala´rio)}(Funciona´rios) retorna o salário médio agrupado por departamento.

Defina as cinco operações principais de álgebra relacional. Define a Junção, Intersecção, Divisão através da utilização das cinco operações básicas.

1. **Seleção (σ):** Retorna as tuplas de uma relação que satisfazem uma condição específica. σcondic¸a˜o(R)\sigma_{\text{condição}}(R)σcondic¸a˜o(R)
2. **Projeção (π):** Extrai colunas específicas de uma relação, eliminando duplicatas. πatributos(R)\pi_{\text{atributos}}(R)πatributos(R)

3. **União (∪):** Combina todas as tuplas de duas relações, eliminando duplicatas. R∪SR \cup SR∪S
4. **Diferença de Conjuntos (−):** Retorna as tuplas de uma relação que não estão presentes em outra. R−SR - SR−S
5. **Produto Cartesiano (×):** Combina cada tupla de uma relação com cada tupla de outra relação. R×SR \times SR×S

Derivadas Utilizando as Operações Básicas

- **Junção (Equijoin):** Junta as tuplas de R e S onde elas têm valores iguais em atributos especificados. Dá-se por um produto carteziano seguido der seleção para filtrar as condições
- **Interseção (∩):** Retorna as tuplas que estão presentes em ambas as relações R e S. R∩S=R−(R−S) da se pela diferença de R com a diferença de R e S.
- **Divisão (÷):** Retorna as tuplas de R que estão associadas a todas as tuplas de S. R÷S=πatributos de R(σcondic¸a˜o de divisa˜o(R×S)) da se pelop prod Cart seguido de seleção da condição da divisão e por fim projeção de atributos de R

6. SQL

O SQL (Structured Query Language) é uma linguagem padrão amplamente adotada para definir e manipular bancos de dados relacionais. Abaixo estão os principais pontos sobre SQL, conforme discutido:

Objetivos do SQL

SQL foi projetado para satisfazer os seguintes objetivos principais:

- **Definição de Dados (DDL):** Permite definir a estrutura do banco de dados, como criar tabelas e definir restrições.
- **Manipulação de Dados (DML):** Facilita a manipulação dos dados dentro das tabelas, incluindo inserção, modificação e exclusão de registros.
- **Consulta (Query):** Permite realizar consultas tanto simples quanto complexas para recuperar dados de forma eficiente.

Importância do SQL

- **Padrão Dominante:** SQL é o único padrão amplamente aceito para linguagens de banco de dados, com suporte de praticamente todos os principais fornecedores.
- **Adoção:** É utilizado em arquiteturas de aplicativos importantes e é um padrão federal nos EUA para sistemas de gerenciamento de banco de dados.

- **Influência:** SQL influencia outros padrões e é usado como ferramenta definicional em várias tecnologias, como IRDS e RDA da ISO.
- **Diversidade:** Existem centenas de produtos baseados em SQL, cada um com seu próprio dialeto, embora o núcleo central da linguagem esteja se padronizando mais.

SELECT

SELECT Especifica quais as colunas que devem aparecer no resultado.
FROM Especifica a/as tabela(s) a ser utilizadas.
WHERE Filtra linhas.
GROUP BY Forma grupos de linhas com o mesmo valor na coluna
HAVING Filtra grupos sujeitos a alguma condição.
ORDER BY Especifica a ordem do resultado.

ORDER BY

A cláusula ORDER BY em SQL permite ordenar os resultados de uma consulta de acordo com critérios específicos. Ela consiste em uma lista de identificadores de coluna que indicam a ordenação desejada, separados por vírgulas. Esses identificadores podem ser nomes de colunas ou números de coluna que representam a posição das colunas na lista SELECT. A ordenação pode ser ascendente (ASC) ou descendente (DESC), e a cláusula ORDER BY deve sempre ser a última cláusula na declaração SELECT.

GROUP BY e HAVING

A cláusula `GROUP BY` no SQL é utilizada para agrupar linhas com valores iguais em colunas especificadas e aplicar funções de agregação a cada grupo, gerando subtotais. Em selects com where e group by, Primeiro, a cláusula `WHERE` filtra as linhas e Depois, `GROUP BY` agrupa as linhas resultantes. `NULLs` são considerados iguais para fins de agrupamento.
A cláusula `HAVING` é usada com `GROUP BY` para filtrar grupos na tabela de resultados final, de forma semelhante ao `WHERE` , mas aplicada aos grupos. A cláusula `HAVING` deve referenciar colunas do `GROUP BY` ou conter funções de agregação, geralmente usadas para condições que não podem ser aplicadas no `WHERE` .

ANY or ALL

As palavras-chave `ANY` e `ALL` são usadas em subconsultas: `ALL` exige que a condição seja satisfeita por todos os valores e `ANY` por qualquer valor. Se a subconsulta estiver vazia, `ALL` retorna verdadeiro e `ANY` falso.

7. SQL: DATA DEFENITION

7.2 Melhorias de integridade

Dados Obrigatórios

Certas colunas devem conter valores válidos, não podendo ser nulas. Isso é implementado com a cláusula `NOT NULL`.

Restrições de Domínio

Cada coluna deve ter um conjunto de valores permitidos. Isso pode ser especificado usando a cláusula `CHECK` ou a instrução `CREATE DOMAIN`, que define domínios explícitos e seus valores aceitáveis.

Integridade de Entidades

A chave primária de uma tabela deve ser única e não nula, garantida pela cláusula `PRIMARY KEY`. Isso assegura que cada registro na tabela é unicamente identificável. Pode-se no entanto garantir valores únicos para chaves alternativas usando UNIQUE

Integridade Referencial

Uma chave estrangeira em uma tabela (tabela filha) deve corresponder a uma chave primária em outra tabela (tabela pai). A integridade referencial é mantida através da cláusula `FOREIGN KEY`, que pode incluir ações de `CASCADE`, `SET NULL`, `SET DEFAULT` ou `NO ACTION` em operações de atualização ou exclusão para garantir a consistência entre as tabelas relacionadas.

1. CASCADE: Deleta a linha da tabela pai e automaticamente deleta as linhas correspondentes na tabela filha, desencadeando essa ação em cascata para outras tabelas que possam ter chaves estrangeiras que referenciam as linhas deletadas.
2. SET NULL: Deleta a linha da tabela pai e define os valores da chave estrangeira na tabela filha como NULL. Essa opção é válida apenas se as colunas de chave estrangeira não forem declaradas como NOT NULL.
3. SET DEFAULT: Deleta a linha da tabela pai e define cada componente da chave estrangeira na tabela filha para o valor padrão especificado. Essa opção é válida apenas se as colunas de chave estrangeira tiverem um valor padrão especificado.
4. NO ACTION: Rejeita a operação de exclusão da tabela pai. Esse é o comportamento padrão se a regra ON DELETE for omitida.

Restrições de Negócio

São regras empresariais aplicadas ao banco de dados, podendo envolver mais de uma tabela. Essas restrições são definidas usando a cláusula `CHECK` ou a instrução `CREATE ASSERTION` para especificar condições complexas que não estão diretamente ligadas a

uma única tabela, como a quantidade máxima de propriedades gerenciadas por um funcionário.

7.3 Data definition

The main SQL data definition language statements are:

CREATE SCHEMA		DROP SCHEMA
CREATE DOMAIN	ALTER DOMAIN	DROP DOMAIN
CREATE TABLE	ALTER TABLE	DROP TABLE
CREATE VIEW		DROP VIEW

- CREATE INDEX: Estrutura que melhora o acesso aos dados de uma tabela baseada nos valores de uma ou mais colunas, podendo ser único para garantir a exclusividade de valores. As colunas são listadas em ordem de importância, podendo ser especificada a ordenação como ascendente (ASC) ou descendente (DESC).

Em resumo, ao utilizar `CREATE UNIQUE INDEX`, você está assegurando que os valores nas colunas indexadas sejam distintos entre si, o que não é garantido ao usar `CREATE INDEX` simples.

Índice Clustered

- **Descrição**: Os dados da tabela são armazenados na mesma ordem do índice. Cada tabela pode ter apenas um índice clustered.
- **Vantagens**: Muito eficiente para consultas de intervalo e para acessar grandes faixas de dados sequencialmente.
- **Desvantagens**: Pode ter impacto em operações de escrita (inserção, atualização, exclusão).
- **Uso Comum**: Consultas que exigem acesso rápido a dados sequenciais.

8. Índice Non-Clustered

- **Descrição**: O índice contém uma cópia das colunas indexadas e um ponteiro para a localização real dos dados na tabela.
- **Vantagens**: Pode haver vários índices non-clustered em uma tabela, e é ideal para melhorar o desempenho de operações de leitura.
- **Desvantagens**: Menos eficiente para leituras sequenciais em comparação com o índice clustered.
- **Uso Comum**: Consultas que precisam buscar dados rapidamente, mas não necessariamente em uma ordem específica.

7.4 Views

Definição de View

- Uma view é o resultado dinâmico de uma ou mais operações relacionais aplicadas às relações base para produzir outra relação.
- É uma relação virtual que não necessariamente existe no banco de dados, mas pode ser produzida sob demanda por um usuário específico no momento da solicitação.

Criando uma View (CREATE VIEW):

- Formato da declaração CREATE VIEW: `CREATE VIEW NomeDaView [(novoNomeColuna [, ... ])] AS subseleção [WITH [CASCADED | LOCAL] CHECK OPTION]`.
- A view é definida especificando uma instrução SELECT SQL.
- É possível atribuir opcionalmente um nome a cada coluna na view.
- Se não forem especificados nomes de coluna, cada coluna na view terá o nome da coluna correspondente na subseleção.
- A subseleção é conhecida como consulta definidora.
- A cláusula WITH CHECK OPTION garante que nenhuma linha que não satisfaça a cláusula WHERE da consulta definidora da view seja adicionada à tabela base subjacente da view.

Removendo uma View (DROP VIEW):
- A view é removida do banco de dados com a declaração DROP VIEW: `DROP VIEW NomeDaView [RESTRICT | CASCADE]`.
- RESTRICT é o comportamento padrão e rejeita a remoção se houver objetos dependentes da view.
- CASCADE remove todas as dependências da view, incluindo outras views definidas sobre ela.

Resolução de Views (View Resolution):
- O processo de mesclar uma consulta em uma view com a consulta definidora da view.
- Substitui nomes de coluna da view na lista SELECT pelos nomes de coluna correspondentes na consulta definidora.
- Substitui nomes de view na cláusula FROM pelas cláusulas FROM correspondentes na consulta definidora.
- Combina a cláusula WHERE da consulta do usuário com a cláusula WHERE da consulta definidora usando o operador lógico AND, entre outros passos.

Restrições em Views:
- Restrições importantes impostas pelo padrão ISO na criação e uso de views.
- Por exemplo, colunas baseadas em funções agregadas só podem aparecer nas cláusulas SELECT e ORDER BY de consultas que acessam a view.

Atualizabilidade de Views:
- Todas as atualizações em uma tabela base são refletidas imediatamente em todas as views que abrangem essa tabela base.
- A view é atualizável se ela satisfizer certos critérios, como não conter funções agregadas na lista SELECT, ter uma única tabela na cláusula FROM, entre outros.

WITH CHECK OPTION:

- Cláusula que impede que uma linha migre para fora da view se ela não satisfizer a cláusula WHERE da consulta definidora da view.

Vantagens e Desvantagens das Views:
- Vantagens: Independência de dados, segurança aprimorada, complexidade reduzida, conveniência, personalização e integridade de dados.
- Desvantagens: Restrição de atualização, restrição de estrutura e desempenho.

Materialização de Views:
- Técnica que armazena uma view como uma tabela temporária no banco de dados para melhorar o desempenho.
- Útil em aplicações onde a taxa de consulta é alta e as views são complexas.

7.5 Transações

Modelo de Transação:

- **Transação:** Unidade lógica de trabalho que envolve uma ou mais instruções SQL.
- **Propriedades ACID:** Assegura que as transações são Atômicas, Consistentes, Isoladas e Duráveis.
  - **Atômica:** Todas as operações são completadas ou nenhuma é.
  - **Consistente:** O estado do banco de dados permanece válido antes e depois da transação.
  - **Isolada:** Mudanças não são visíveis para outras transações até a conclusão.
  - **Durável:** As mudanças persistem mesmo após falhas do sistema.
- **Início Automático:** Inicia com a primeira instrução SQL executada (ex.: SELECT, INSERT).

Conclusão da Transação:

- **COMMIT:** Finaliza e salva permanentemente as mudanças.
- **ROLLBACK:** Reverte todas as mudanças feitas pela transação.
- **SQL Programático:** Terminação normal finaliza com sucesso; terminação anormal aborta a transação.

Configuração de Transações:

- **SET TRANSACTION:** Define aspectos como nível de leitura e isolamento.
  - **READ ONLY/WRITE:** Define se a transação é somente leitura ou leitura e escrita.
  - **Níveis de Isolamento:** READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE (apenas SERIALIZABLE garante cronogramas serializáveis).

Verificação de Restrições:

- **Restrições Diferidas:** Permite verificar restrições de integridade no commit da transação, em vez de após cada instrução SQL.
- **SET CONSTRAINTS:** Define o modo de verificação (IMEDIATO ou DIFERIDO) para as restrições.

7.6 Mecanismos de Controle de Acesso em SQL

1. Controle de Acesso Discricionário (DAC):

- **Autorização:** Cada usuário recebe privilégios específicos sobre objetos de banco de dados, como tabelas e visões.
- **Privilégios:** Permitem ações como selecionar (SELECT), inserir (INSERT), atualizar (UPDATE), e excluir (DELETE) dados.
- **Comandos:**
  - **GRANT:** Concede privilégios a usuários específicos.
  - **REVOKE:** Revoga privilégios previamente concedidos.
- **Flexibilidade:** Usuários com privilégios podem compartilhá-los com outros, mas isso pode ser vulnerável a abusos se um usuário autorizado for enganado.

2. Controle de Acesso Obrigatório (MAC):

- **Classificação:** Objetos de banco de dados são atribuídos a níveis de segurança (como Confidencial, Secreto).
- **Controle Rígido:** Cada usuário precisa ter a autorização adequada para acessar dados de acordo com sua classificação.
- **Exemplo de Modelo:** Bell-LaPadula, usado para garantir que informações sensíveis sejam acessadas apenas por usuários com a devida autorização.

Em resumo, o SQL utiliza principalmente o controle de acesso discricionário, onde os usuários controlam quem pode acessar os dados, mas também é possível implementar controle mais rígido usando modelos de segurança obrigatórios para proteger informações sensíveis.

8 . Advanced SQL

8.1.1 Declarações

- **Variáveis e Constantes:** Devem ser declaradas antes de serem usadas. Exemplos:
  - `vStaffNo VARCHAR2(5);`
  - `MAX_PROPERTIES CONSTANT NUMBER := 100;`
- **Tipo de Dados:** Variáveis podem ser do mesmo tipo de uma coluna de uma tabela usando `%TYPE` ou de uma linha inteira com `%ROWTYPE`.
  - `vStaffNo Staff.staffNo%TYPE;`
  - `vStaffRec Staff%ROWTYPE;`
- **Atribuições NOT NULL:** Variáveis `NOT NULL` requerem um valor inicial.

8.1.2 Atribuições

- **Atribuições Simples:** Usam o operador `:=`.
  - `vStaffNo := 'SG14';`

- **Atribuições por Consulta SQL:** Usam `SELECT INTO`.
  - `SELECT COUNT(*) INTO x FROM PropertyForRent WHERE staffNo = vStaffNo;`

8.1.3 Instruções de Controle

- **Condicionais (IF):** Permitem executar código com base em condições.
  - `IF (condition) THEN ... ELSE ... END IF;`
- **Condicionais (CASE):** Escolhem caminhos de execução com base em alternativas.
  - `CASE (operand) WHEN ... THEN ... ELSE ... END CASE;`
- **Laços (LOOP):** Executam código repetidamente até uma condição ser satisfeita.
  - `LOOP ... EXIT WHEN (condition) END LOOP;`
- **Laços (WHILE):** Continuam enquanto uma condição for verdadeira.
  - `WHILE (condition) LOOP ... END LOOP;`
- **Laços (FOR):** Iteram um número definido de vezes.
  - `FOR indexVariable IN lowerBound .. upperBound LOOP ... END LOOP;`

8.1.4 Exceções em PL/SQL

- **Exceções:** Identificadores que interrompem a execução de um bloco quando um erro ocorre.
  - **Automáticas:** Ex.: `NO_DATA_FOUND` é levantada quando nenhuma linha é retornada.
  - **Usuário:** Ex.: Uso de `RAISE` para levantar exceções definidas pelo usuário.
- **Manipulação de Exceções:** Utiliza `DECLARE HANDLER` para definir ações para exceções específicas.

8.1.5 Cursores em PL/SQL

- **Definição de Cursor:** Usado para manipular múltiplas linhas de resultado de uma consulta.
  - `CURSOR propertyCursor IS SELECT ...`
- **Operações de Cursor:** Abrir (`OPEN`), buscar (`FETCH`), e fechar (`CLOSE`).
- **Cursores com Parâmetros:** Permitem reusar a definição de cursor com diferentes critérios.
  - `CURSOR propertyCursor (vStaffNo VARCHAR2) IS SELECT ...`
- **Atualização com Cursores:** Permite atualizar ou excluir linhas após a busca com `FOR UPDATE`.
  - `UPDATE PropertyForRent SET staffNo = 'SG37' WHERE CURRENT OF propertyCursor;`

8.2 Subprogramas, Procedimentos Armazenados, Funções

Subprogramas

- **Blocos PL/SQL Nomeados:** Podem receber parâmetros e ser invocados.

    - **Procedures (Procedimentos):** Executam ações e podem modificar e retornar dados, mas não retornam um valor específico.
    - **Functions (Funções):** Sempre retornam um único valor para o chamador.

Parâmetros

- **Tipos de Parâmetros:**
    - **IN:** Apenas entrada.
    - **OUT:** Apenas saída.
    - **IN OUT:** Entrada e saída.

Vantagens

- **Modularidade e Extensibilidade:** Facilitam a organização e a manutenção do código.
- **Reutilização e Manutenção:** Promovem a reutilização de código e facilitam a manutenção.
- **Abstração:** Permitem a abstração de operações complexas.

Pacotes

- **Definição:** Conjunto de procedures, functions, variáveis e declarações SQL agrupadas como uma unidade de programa.
- **Estrutura de Pacote:**
    - **Especificação (Specification):** Declara os componentes públicos.
    - **Corpo (Body):** Define e implementa todos os componentes (públicos e privados).
- **Benefícios:** Oferecem encapsulamento e organização.

8.3 Triggers

Definição

- **Ação Automatizada:** Executada quando um evento específico ocorre no banco de dados.

Usos Comuns

- **Integridade Referencial:** Garantir a consistência dos dados.
- **Restrições Complexas:** Aplicar regras complexas de validação.
- **Auditoria:** Monitorar e registrar alterações nos dados.

Estrutura Básica

```
CREATE TRIGGER TriggerName
BEFORE | AFTER | INSTEAD OF
INSERT | DELETE | UPDATE [OF TriggerColumnList]
ON TableName
[REFERENCING {OLD | NEW} AS {OldName | NewName}]
[FOR EACH {ROW | STATEMENT}]
[WHEN Condition]
<trigger action>
```

Modelo Evento-Condição-Ação (ECA)

- **Evento:** Operação que dispara o trigger (ex.: INSERT, UPDATE, DELETE).

- **Condição:** Opcional; se especificada, a ação é executada apenas se a condição for verdadeira.
- **Ação:** Código SQL executado quando o evento ocorre e a condição é verdadeira.

Tipos de Triggers

- **Row-Level (Por Linha):** Executa para cada linha afetada pelo evento.
- **Statement-Level (Por Declaração):** Executa uma vez para o evento, independente do número de linhas afetadas.
- **INSTEAD OF:** Substitui a execução de uma operação SQL em views não modificáveis diretamente.

Vantagens de Uso de Triggers

- Automatização de Tarefas
- Manutenção da Integridade
- Auditoria e Monitoramento
- Redução de Redundância

Desvantagens de Uso de Triggers

- Complexidade de Depuração
- impacto no Desempenho
- Manutenção complexa
- Ordem de Execução não garantida

10 . Database System Development Lifecycle

Ciclo de Vida do Sistema de Banco de Dados (DSDLC) é uma abordagem estruturada para o desenvolvimento de sistemas de banco de dados, essenciais para o sistema de informação de uma organização. Esse ciclo é composto por várias etapas que garantem a criação eficiente e eficaz do banco de dados, atendendo tanto às necessidades organizacionais quanto às técnicas.

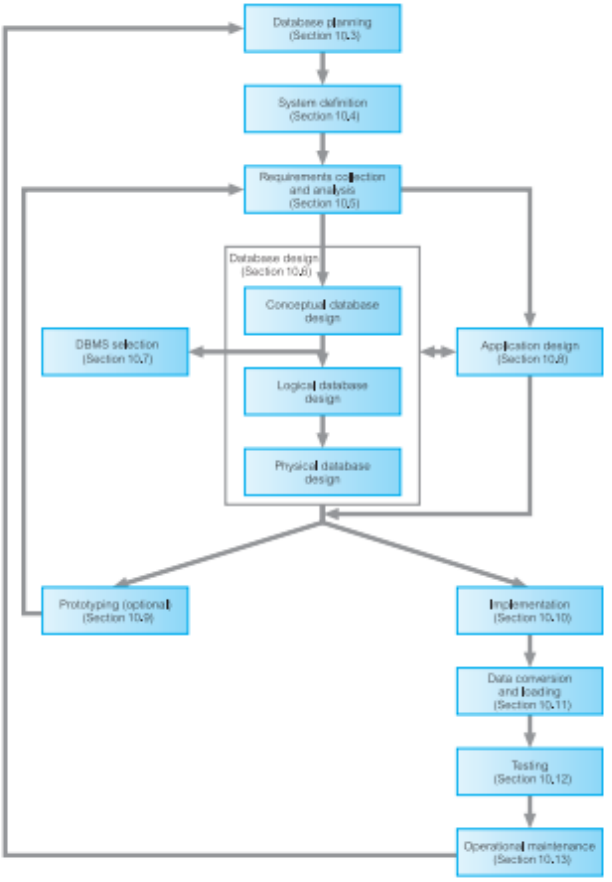


Figure 10.1 The stages of the database system development lifecycle.

Principais Etapas do Ciclo de Vida do Sistema de Banco de Dados:**

1. **Planejamento de Banco de Dados (Seção 10.3)**:
   - **Objetivo**: Integrar o desenvolvimento do banco de dados à estratégia geral do sistema de informação da organização.
   - **Atividades Principais**:
     - Definição da missão e objetivos do sistema de banco de dados.
     - Desenvolvimento de normas para coleta e formato de dados, além da documentação.

2. **Definição do Sistema (Seção 10.4)**:
   - **Objetivo**: Especificar o escopo e os limites do sistema de banco de dados.
   - **Atividades Principais**:
     - Identificação das principais visões dos usuários e áreas de aplicação.
     - Determinação dos requisitos e restrições do sistema.

3. **Coleta e Análise de Requisitos (Seção 10.5)**:
   - **Objetivo**: Coletar e analisar as necessidades dos usuários e do sistema.

4. **design de Banco de Dados (Seção 10.6)**:
   - **Objetivo**: Criar o design conceitual, lógico e físico do banco de dados.

- **Atividades Principais**:
  - Projeto conceitual: definição dos principais conceitos e relações do banco de dados.
  - Projeto lógico: estruturação dos dados de acordo com os modelos lógicos.
  - Projeto físico: especificação da organização física dos dados no armazenamento.

5. **Seleção do SGBD (Seção 10.7)**:
   - **Objetivo**: Escolher um Sistema de Gerenciamento de Banco de Dados (SGBD) adequado para o sistema.
   - **Atividades Principais**:
     - Avaliar e selecionar o SGBD que melhor atende às necessidades do projeto.

6. **design de Aplicação (Seção 10.8)**:
   - **Objetivo**: Desenvolver a interface do usuário e os programas de aplicação que utilizarão e processarão o banco de dados.
   - **Atividades Principais**:
     - Projetar a interface de usuário e os programas que interagem com o banco de dados.

7. **Prototipagem (opcional) (Seção 10.9)**:
   - **Objetivo**: Construir um modelo funcional do banco de dados para visualização e avaliação.
   - **Atividades Principais**:
     - Criar um protótipo para testar e ajustar o design antes da implementação final.

8. **Implementação (Seção 10.10)**:
   - **Objetivo**: Criar as definições físicas do banco de dados e os programas de aplicação.
   - **Atividades Principais**:
     - Implementar o banco de dados e desenvolver os programas necessários.

9. **Conversão e Carregamento de Dados (Seção 10.11)**:
   - **Objetivo**: Carregar dados do sistema antigo para o novo e converter aplicações existentes.
   - **Atividades Principais**:
     - Transferir dados e converter aplicações para o novo sistema.

10. **Testes (Seção 10.12)**:
    - **Objetivo**: Testar o sistema de banco de dados para detectar erros e validar os requisitos.
    - **Atividades Principais**:
      - Realizar testes para assegurar que o banco de dados atende aos requisitos especificados.

11. **Manutenção Operacional (Seção 10.13)**:
    - **Objetivo**: Monitorar e manter o sistema de banco de dados em operação.
    - **Atividades Principais**:
      - Manter o banco de dados, monitorar o desempenho e incorporar novos requisitos conforme necessário.

O ciclo de vida de um sistema de banco de dados é crucial para o desenvolvimento de sistemas eficazes e eficientes. Cada etapa do DSDLC desempenha um papel vital na criação de um banco de dados que atende às necessidades operacionais e estratégicas de uma organização, garantindo a sua integridade, desempenho e capacidade de evolução conforme novas demandas surgem.

Abordagens de Gestão de Requisitos na Coleta e Análise de Requisitos

Durante a coleta e análise de requisitos para um sistema de banco de dados, existem três principais abordagens para gerenciar as diferentes visões de usuários:

1. **Abordagem Centralizada (Centralized Approach)**:
   - **Descrição**: Requisitos de todas as visões de usuários são combinados em uma única lista.
   - ou seja, unifica as necessidades de diferentes usuários em um único conjunto de requisitos, adequado quando há muita sobreposição de necessidades.
   - **Aplicação**: Usada quando há significativa sobreposição de requisitos entre as visões dos usuários.
   - **Resultado**: Criação de um modelo de dados global que representa todas as visões dos usuários.
2. **Abordagem de Integração de Visões (View Integration Approach)**:
   - **Descrição**: Mantém os requisitos de cada visão de usuário em listas separadas.
   - **Aplicação**: Preferida quando há diferenças significativas entre as visões dos usuários e o sistema é complexo.
   - **Resultado**: Cada visão de usuário é inicialmente modelada separadamente e posteriormente integrada em um modelo de dados global.
3. **Combinação das Abordagens (Combination Approach)**:
   - **Descrição**: Utiliza uma mistura das abordagens centralizada e de integração de visões.
   - **Aplicação**: Adequada para sistemas complexos, onde algumas visões podem ser combinadas antes de serem integradas com outras visões.
   - **Resultado**: Requisitos de algumas visões são centralizados para formar um modelo local, que é então integrado com outros modelos locais para formar um modelo de dados global.

Fases do Design de Banco de Dados:

1. **Design Conceitual**:
   - **Objetivo**: Criar um esboço geral do banco de dados que mostra as principais informações e suas relações, sem considerar como será implementado.
   - **Ferramentas**: Modelos como Entidade-Relacionamento (ER), que mapeiam entidades (coisas ou conceitos importantes) e como elas se relacionam.
2. **Design Lógico**:
   - **Objetivo**: Transformar o esboço em um modelo mais detalhado, adequado ao tipo de banco de dados que será usado, como o modelo relacional.
   - **Ferramentas**: Normalização, que organiza os dados para evitar duplicações e inconsistências.
3. **Design Físico**:
   - **Objetivo**: Planejar a implementação real do banco de dados em um sistema específico, definindo como os dados serão armazenados e acessados.
   - **Ferramentas**: Estruturas de armazenamento, métodos de acesso e medidas de segurança para otimizar o desempenho.

Modelagem de Dados:

- **Objetivo**: Compreender e comunicar as necessidades de informação da empresa.
- **Características de um Bom Modelo**:
  - **Clareza**: Deve ser fácil de entender e consistente com a organização.
  - **Eficácia**: Sem dados duplicados e deve permitir evolução futura.
  - **Utilização**: Representar de forma clara os dados e suas relações, usando diagramas simples.

Importância da Integração e Continuidade:

- O design do banco de dados deve ser flexível e adaptável a mudanças.
- É fundamental para o sucesso do sistema e deve refletir a realidade da empresa para facilitar futuras atualizações e garantir um bom desempenho.

Stage of database system development lifecycle	Examples of data captured	Examples of documentation produced
Database planning	Aims and objectives of database project	Mission statement and objectives of database system
System definition	Description of major user views (includes job roles or business application areas)	Definition of scope and boundary of database application; definition of user views to be supported
Requirements collection and analysis	Requirements for user views; systems specifications, including performance and security requirements	Users' and system requirements specifications
Database design	Users' responses to checking the logical database design; functionality provided by target DBMS	Conceptual/logical database design (includes ER model(s), data dictionary, and relational schema); physical database design
Application design	Users' responses to checking interface design	Application design (includes description of programs and user interface)
DBMS selection	Functionality provided by target DBMS	DBMS evaluation and recommendations
Prototyping	Users' responses to prototype	Modified users' requirements and systems specifications
Implementation	Functionality provided by target DBMS	
Data conversion and loading	Format of current data; data import capabilities of target DBMS	
Testing	Test results	Testing strategies used; analysis of test results
Operational maintenance	Performance testing results; new or changing user and system requirements	User manual; analysis of performance results; modified users' requirements and systems specifications

Técnicas de Coleta de Dados no Ciclo de Vida de Desenvolvimento de Sistemas de Banco de Dados

1. Análise da Documentação

- **Objetivo**: Obter uma visão geral sobre a necessidade de um banco de dados, compreendendo o problema e a parte da empresa afetada.
- **Tipos de Documentação**: Memorandos internos, relatórios de desempenho, fluxogramas, manuais de usuário, e documentos de sistemas atuais.

2. Entrevistas

- **Objetivo**: Coletar informações diretamente dos usuários ou partes interessadas.
- **Tipos**: Entrevistas não estruturadas (mais flexíveis, mas podem perder o foco) e estruturadas (mais direcionadas, com perguntas específicas).
- **Vantagens**: Permite respostas detalhadas e personalizadas, promove o engajamento dos entrevistados.
- **Desvantagens**: Consome tempo, pode ser caro e depende das habilidades de comunicação do entrevistador.

3. Observando o Funcionamento da Organização

- **Objetivo**: Entender como as atividades são realmente realizadas no ambiente de trabalho.
- **Método**: Observação direta das operações e tarefas diárias dos usuários.
- **Vantagens**: Valida dados coletados por outros métodos, fornece uma visão realista das operações.
- **Desvantagens**: As pessoas podem agir de forma diferente quando estão sendo observadas, o que pode levar a uma coleta de dados enviesada.

4. Pesquisa

- **Objetivo**: Encontrar informações adicionais e soluções para problemas semelhantes.
- **Fontes**: Revistas especializadas, livros de referência, internet, grupos de usuários e fóruns.
- **Vantagens**: Pode economizar tempo se soluções existentes forem encontradas e ajuda a atualizar o conhecimento do pesquisador.
- **Desvantagens**: Pode ser difícil encontrar informações relevantes ou específicas para o problema em questão.

5. Questionários

- **Objetivo**: Coletar dados de um grande número de pessoas de maneira eficiente.
- **Tipos**: Perguntas de formato livre (permitem respostas detalhadas) e perguntas de formato fixo (respostas específicas, facilitando a tabulação).
- **Vantagens**: Eficiente para coletar uma grande quantidade de dados de muitos participantes.
- **Desvantagens**: Pode limitar as respostas e não capturar informações detalhadas que podem ser úteis.

Comparação das Técnicas de Coleta de Dados

Técnica	Vantagens	Desvantagens
Análise da Documentação	Acesso a informações detalhadas e históricas	Pode estar desatualizada e ser de difícil interpretação
Entrevistas	Respostas detalhadas, personalizadas, engajamento	Consome tempo, caro, depende de boas habilidades de comunicação
Observação	Validação de dados, visão realista das operações	Comportamento pode mudar sob observação, coleta de dados enviesada
Pesquisa	Economiza tempo com soluções existentes, atualiza conhecimento	Difícil encontrar informações relevantes ou específicas
Questionários	Coleta eficiente de dados de muitos participantes	Respostas limitadas, pode não capturar detalhes importantes

12 . Entity-Relationship Modeling

12.1 Tipos de Entidade

- **Definição**: Grupo de objetos com propriedades comuns, que têm existência independente e podem ser físicos ou conceituais.
- fisicos pode ser pessoas, propriedade, carro... e conceptuais pode ser algum evento que se dá geralmente pela relação de entidades fisicas

Physical existence	
Staff	Part
Property	Supplier
Customer	Product
Conceptual existence	
Viewing	Sale
Inspection	Work experience

- **Ocorrência de Entidade**: Objeto único identificável de um tipo de entidade.
- **Representação**: Cada tipo de entidade é mostrado como um retângulo com o nome da entidade.

12.2 Tipos de Relacionamento

- **Definição**: Conjunto de associações significativas entre tipos de entidade.
- **Ocorrência de Relacionamento**: Associação única entre ocorrências de entidades participantes.
- **Representação**: As ocorrências de relacionamento são representadas por linhas conectando as entidades.

12.2.1 Grau do Tipo de Relacionamento

- **Grau de Relacionamento**: Refere-se ao número de tipos de entidades envolvidos em um relacionamento.
- **Tipos de Graus**:
  - **Binário**: Inclui dois tipos de entidade. É o mais comum.
  - **Ternário**: Envolve três tipos de entidade.
  - **Quaternário**: Inclui quatro tipos de entidade.
- **Representação Diagramática**: Em UML, relacionamentos de grau superior ao binário são representados por um losango com o nome do relacionamento no interior.

12.2.2 Relacionamento Recursivo

- **Definição**: Tipo de relacionamento onde o mesmo tipo de entidade participa mais de uma vez, em diferentes papéis.
- **Exemplo**: A entidade Staff pode ser tanto Supervisor quanto Supervisionado em um relacionamento chamado Supervises.
- **Nomes de Papel**: Usados para indicar a função de cada participante no relacionamento, especialmente útil para relacionamentos recursivos.

12.3 Atributos

- **Definição**: Propriedades de um tipo de entidade ou de um tipo de relacionamento.
- **Domínio do Atributo**: Conjunto de valores permitidos para um atributo.
- **Classificação de Atributos**:
  - **Simples**: Não pode ser subdividido.

    - **Composto**: Pode ser subdividido em componentes menores.
    - **Univalorado**: Contém um único valor por ocorrência de entidade.
    - **Multivalorado**: Pode conter múltiplos valores por ocorrência de entidade.
    - **Derivado**: Valor calculado a partir de outros atributos, que podem estar na mesma ou em diferentes entidades..
- Representação Diagramática dos Atributos
    - **Simples e Univalorados**: Listados abaixo do nome da entidade.
    - **Compostos**: Listados com seus componentes abaixo e indentados.
    - **Multivalorados**: Indicados com o intervalo de valores possível.
    - **Derivados**: Prefixados com uma barra "/".

12.3.4 Chaves

- **Chave Candidata**: Conjunto mínimo de atributos que identifica de forma única cada ocorrência de uma entidade.
- **Chave Primária**: Chave candidata selecionada para identificar de forma única cada ocorrência de uma entidade.
- **Chave Composta**: Chave candidata que consiste em dois ou mais atributos.

12.4 Strong and Weak Entity Types

- **Entidade Forte (Strong Entity Type)**:
    - **Definição**: Não depende da existência de outra entidade para existir.
    - **Identificação**: Cada ocorrência é identificada de forma única por uma chave primária.
- **Entidade Fraca (Weak Entity Type)**:
    - **Definição**: Depende da existência de outra entidade para existir.
    - **Identificação**: Não pode ser identificada de forma única apenas pelos seus atributos; depende de uma entidade "dona".
    - **Exemplo**: A entidade Preference não pode ser identificada apenas pelos seus atributos e depende da entidade Client para identificação.

Cardinalidade/multiplicidade de relacionamemntos

Restrições de Cardinalidade e Participação

- **Cardinalidade**:
    - Define o número máximo de ocorrências de relacionamento para uma entidade.
        - "1..1": Exatamente uma ocorrência.
        - "0..*": Zero ou muitas ocorrências.
- **Participação**:
    - Descreve se todas (obrigatória) ou apenas algumas (opcional) ocorrências de entidade participam de um relacionamento.

- Valores mínimos como "0" (opcional) ou "1" (obrigatório).

Tabela Resumo para Restrições de Multiplicidade

Representação	Significado
0..1	Zero ou uma ocorrência de entidade
1..1	Exatamente uma ocorrência de entidade
0..*	Zero ou muitas ocorrências de entidade
1..*	Uma ou muitas ocorrências de entidade
5..10	Mínimo de 5 até um máximo de 10
0, 3, 6–8	Zero ou exatamente 3 ou 6 a 8 ocorrências

Relacionamentos Complexos

- **Relacionamentos Complexos (mais do que Binários)**:
  - **Exemplo**: Um relacionamento ternário onde um `Staff` registra um `Client` em uma `Branch`.
  - **Representação de Multiplicidade**: Considere `Staff` e `Branch` fixos, depois determine o número possível de entidades `Client` envolvidas.
    - **Representação**: Em um relacionamento ternário, a multiplicidade é representada para cada par de entidades fixas em relação à terceira entidade.
  - **Diagrama ER**:
    - `0..*` para `Client` quando `Staff` e `Branch` são fixos.
    - `1..1` para `Branch` quando `Staff` e `Client` são fixos.
    - `1..1` para `Staff` quando `Client` e `Branch` são fixos.

13.1 Specialization/Generalization

- **Superclasse**: É uma entidade geral que contém um grupo de entidades mais específicas, conhecidas como subclasses.
- **Subclasse**: É uma entidade mais específica que herda atributos e relacionamentos de uma superclasse e pode ter características adicionais próprias.
- **Herança de Atributos**: Subclasses herdam atributos da superclasse, além de terem atributos específicos próprios.

- **Relacionamento Superclasse/Subclasse**: Cada membro de uma subclasse também é membro da superclasse. Existe um relacionamento 1:1 entre superclasse e subclasses.

2. Processo de Especialização

- **Definição**: Processo top-down para identificar características distintivas das entidades e criar subclasses baseadas nessas características.
- **Objetivo**: Maximizar as diferenças entre membros de uma entidade.
- **Exemplo**: Especialização de Staff em Manager, SalesPersonnel, e Secretary com atributos específicos para cada tipo.

3. Processo de Generalização

- **Definição**: Processo bottom-up para identificar características comuns entre entidades e agrupá-las sob uma superclasse.
- **Objetivo**: Minimizar as diferenças entre entidades.
- **Exemplo**: Generalização de Manager, SalesPersonnel, e Secretary sob a superclasse Staff.

4. Representação Diagramática

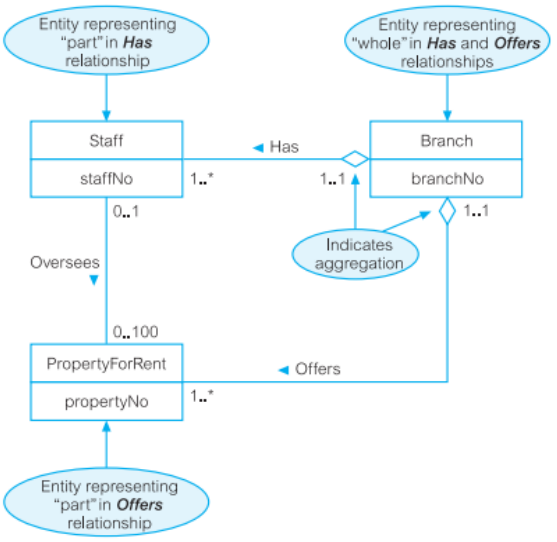
- **Notação UML**: Usa-se um triângulo apontando para a superclasse para indicar especialização/generalização.
- **Atributos**: Atributos específicos de subclasses são listados dentro das caixas das subclasses.

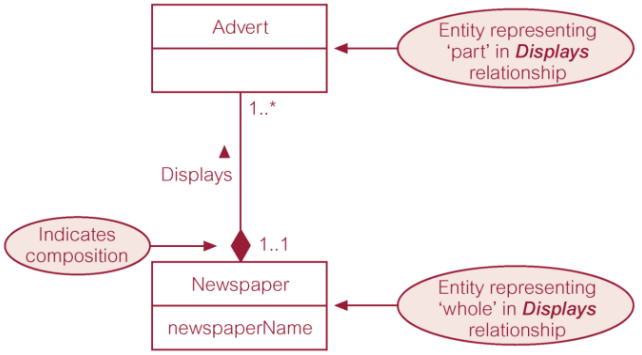
5. Restrições de Especialização/Generalização

- **Participação**: determina se todos os membros da superclasse devem ser membros de uma subclasse. Pode ser obrigatória ou opcional.
- **Disjunção**: descreve a relação entre as subclasses indicando se um membro de uma subclasse pode ser membro de outras. Pode ser disjunta (um membro só pode pertencer a uma subclasse) ou não disjunta (um membro pode pertencer a múltiplas subclasses).

Agregação vs Composicao

Agregação: Relacionamento onde uma entidade maior (o "todo") é composta por entidades menores (as "partes"), como um departamento que possui funcionários.
Composição: Tipo de agregação com forte vínculo e ciclo de vidacomum entre o "todo" e as "partes", como um jornal que contém anúncios específicos.





14. Normalização

Normalização é uma técnica formal para analisar e organizar relações em um banco de dados com base em suas chaves primárias e dependências funcionais. Ela segue uma série de regras para testar e ajustar relações, garantindo um banco de dados mais eficiente e menos sujeito a anomalias.

Objetivos da Normalização

A normalização é um processo essencial no design de bancos de dados, cujo principal objetivo é organizar os dados de maneira eficiente para evitar problemas comuns como redundância e inconsistência. Os principais objetivos da normalização são:

1. **Redução de Redundância**- Minimiza a duplicação de dados, economizando espaço de armazenamento.
2. **Eliminação de Anomalias de Atualização**: Facilita a atualização consistente dos dados, evitando inconsistências.
3. **Facilitação da Manutenção de Dados**:Simplifica a adição, modificação e exclusão de informações.

4. **Melhoria na Integridade dos Dados**:Garante precisão e consistência, mantendo a integridade referencial.
5. **Optimização do Armazenamento**:Utiliza o espaço de armazenamento de forma eficiente.
6. **Aprimoramento do Desempenho de Consultas**:Estrutura organizada que facilita consultas mais eficientes.
7. **Facilitação da Expansão e Modificação do Esquema**:Permite modificações e expansões com menor impacto na consistência dos dados.

Processo de Normalização

A normalização é realizada em etapas, cada uma correspondendo a uma forma normal. Começa-se com uma tabela não normalizada, chamada de **Forma Não Normalizada (UNF)**, e a cada etapa, a tabela é transformada para atender aos requisitos das formas normais subsequentes, até alcançar pelo menos a 3NF para evitar anomalias de atualização.

FORMAS nORMAIS

Primeira Forma Normal (1FN)

- **Definição**: Assegura que todos os valores em uma tabela sejam atômicos, ou seja, indivisíveis.
- **Objetivo**: Eliminar grupos de valores repetidos e garantir que cada célula de uma tabela contenha um valor único e indivisível por linha.

Segunda Forma Normal (2FN)

- **Definição**: Uma tabela está na 2FN se ela estiver na 1FN e todos os atributos não-chave forem completamente dependentes da chave primária.
- **Objetivo**: Garantir que todos os atributos não-chave dependam da chave primária inteira, eliminando dependências parciais (isto é, a dependência de apenas uma parte da chave primária, caso esta seja composta).

Terceira Forma Normal (3FN)

- **Definição**: Uma tabela está na 3FN se ela estiver na 2FN e todos os atributos não-chave forem dependentes apenas da chave primária, não de outros atributos não-chave.
- **Objetivo**: Eliminar dependências transitivas, onde um atributo não-chave depende de outro atributo não-chave, promovendo uma maior independência entre os dados.

Quarta Forma Normal (4FN)

- **Definição**: Uma tabela está na 4FN se estiver na 3FN e não tiver dependências multivaloradas. Dependências multivaloradas ocorrem quando um atributo em uma tabela está relacionado a múltiplos valores de outro atributo de maneira independente.

- **Objetivo**: Eliminar dependências multivaloradas, garantindo que cada par de atributos não esteja relacionado de maneira multivalorada, o que evita a redundância e complexidade desnecessária.

DATA WAREHOUSE

Um **Data Warehouse** é um sistema de gerenciamento de dados projetado para armazenar grandes volumes de dados integrados, provenientes de várias fontes, com o objetivo de apoiar a tomada de decisões empresariais. Ele é usado para coletar, organizar, e disponibilizar dados históricos de uma maneira que facilite a análise e a geração de relatórios.

bENEFICIOS
A implementação bem-sucedida de um Data Warehouse pode trazer vários benefícios importantes para uma organização:

1. **Alto Retorno sobre Investimento (ROI)**:
   - Apesar dos custos significativos para implementar um Data Warehouse, estudos mostram um ROI médio de 401% em três anos, destacando os altos retornos potenciais.
2. **Vantagem Competitiva**:
   - Proporciona acesso a dados que revelam informações valiosas sobre clientes, tendências e demandas, conferindo uma vantagem sobre a concorrência.
3. **Aumento da Produtividade dos Tomadores de Decisões**:
   - Facilita a análise precisa e consistente de dados históricos e integrados, melhorando a qualidade e a eficiência nas decisões empresariais.

- **Facilidade de Análise de Tendências**:

Comparação entre Sistemas OLTP e Data Warehousing

Os sistemas OLTP (Online Transaction Processing) e Data Warehousing são projetados para diferentes propósitos:

- **OLTP**:
  - Suporta processamento operacional.
  - Dados atuais, detalhados e com alta frequência de transações previsíveis.
  - Serve a um grande número de usuários operacionais.
- **Data Warehouse**:
  - Suporta processamento analítico.
  - Dados históricos e sumarizados, com baixo volume de transações imprevisíveis.

- Serve a um número menor de usuários, principalmente gerenciais e analíticos.

Problemas Associados à Data Warehousing

1. **Subestimação de Recursos para ETL**:
   - O tempo necessário para extrair, transformar e carregar dados é frequentemente subestimado, exigindo muitos recursos.
2. **Problemas Ocultos nos Sistemas de Origem**:
   - Problemas não detectados podem surgir ao integrar dados de vários sistemas, exigindo correção tanto nos sistemas de origem quanto no Data Warehouse.
3. **Dados Necessários Não Capturados**:
   - Pode haver uma necessidade de dados que os sistemas atuais não capturam, exigindo modificações nos sistemas existentes.
4. **Aumento das Demandas dos Usuários**:
   - Com o acesso a novas ferramentas de consulta e relatório, as demandas dos usuários por suporte tendem a aumentar.
5. **Homogeneização de Dados**:
   - A integração de dados de diferentes áreas pode levar à perda de detalhes específicos e valiosos.
6. **Alta Demanda por Recursos**:
   - Requer grandes quantidades de espaço em disco e outros recursos para armazenar e gerenciar dados.
7. **Propriedade dos Dados**:
   - Pode haver mudanças na percepção de propriedade dos dados dentro da organização, com dados antes restritos sendo disponibilizados a um público mais amplo.
8. **Alta Manutenção**:
   - Requer manutenção constante para se alinhar às mudanças nos processos e sistemas da organização.
9. **Projetos de Longa Duração**:
   - A construção de um Data Warehouse pode levar vários anos, levando algumas organizações a optarem por data marts menores e mais rápidos de implementar.
10. **Complexidade de Integração**:
    - A integração de várias ferramentas de Data Warehousing pode ser complexa e requer um planejamento cuidadoso para garantir a eficácia do sistema.

31.2 Data Warehouse Architecture

1. **Operational Data (Dados Operacionais)**:
   a. Origem inicial dos dados para o Data Warehouse, vindo de várias fontes como mainframes, bancos de dados relacionais, e sistemas externos.

2. **Operational Data Store (ODS - Repositório de Dados Operacionais)**:
    a. Armazém de dados operacionais integrados e atuais usado como preparação antes de serem carregados no Data Warehouse.
3. **ETL Manager (Gerenciador de ETL)**:
    a. Responsável por extrair, transformar e carregar os dados no Data Warehouse, normalmente integrando dados do ODS ou diretamente das fontes.
4. **Warehouse Manager (Gerenciador de Data Warehouse)**:
    a. Gerencia os dados no Data Warehouse, incluindo transformações, fusões, criação de índices e agregações, além de backup e arquivamento.
5. **Query Manager (Gerenciador de Consultas)**:
    a. Direciona consultas dos usuários para as tabelas apropriadas no Data Warehouse e agenda execuções para otimização de desempenho.
6. **Detailed Data (Dados Detalhados)**:
    a. Armazena todos os dados detalhados conforme o esquema do Data Warehouse.
7. **Lightly and Highly Summarized Data (Dados Levemente e Altamente Sumarizados)**:
    a. Armazena dados sumarizados para acelerar o desempenho das consultas, atualizados regularmente para refletir mudanças nos perfis de consulta.
8. **Archive/Backup Data (Dados de Arquivo/Backup)**:
    a. Mantém dados detalhados e sumarizados para fins de arquivamento e backup, incluindo dados sumarizados mantidos online além do período de dados detalhados.
9. **Metadata (Metadados)**:
    a. Armazena definições usadas pelos processos do Data Warehouse, como mapeamento de fontes de dados e automação de processos de gestão.
10. **End-User Access Tools (Ferramentas de Acesso para Usuários Finais)**:
    a. Utilizadas por usuários finais para interagir com o Data Warehouse, incluindo ferramentas de relatórios, consultas, desenvolvimento de aplicativos, OLAP e mineração de dados.

TABLE 31.3 The requirements for a data warehouse DBMS.

Load performance
Load processing
Data quality management
Query performance
Terabyte scalability
Mass user scalability
Networked data warehouse
Warehouse administration
Integrated dimensional analysis
Advanced query functionality

Data MArt

Um Data Mart é um subconjunto de dados corporativos projetado para atender aos requisitos analíticos de um grupo específico de usuários ou de um departamento dentro de uma organização. Aqui estão os pontos principais sobre Data Marts:

1. **Definição**:
   - Um Data Mart é um banco de dados que contém uma parte dos dados corporativos, focado em suportar as necessidades analíticas de um departamento ou grupo de usuários com requisitos semelhantes.
2. **Metodologias de Construção**:
   - **Metodologia de Kimball**: Um Data Mart em metodologia Kimball é a implementação física de um esquema estelar (star schema) modelado em torno de um processo de negócio específico. Vários Data Marts de Kimball podem ser integrados para formar um Data Warehouse Empresarial.
   - **Metodologia de Inmon**: Um Data Mart na metodologia de Inmon é a implementação física de um banco de dados que suporta os requisitos analíticos de uma unidade de negócio específica. O Data Mart de Inmon recebe dados do Data Warehouse Empresarial.
3. **Propósitos de Criação**:
   - Fornecer aos usuários acesso aos dados que eles mais frequentemente analisam.
   - Estruturar os dados de forma que corresponda à visão coletiva de um grupo de usuários em um departamento ou interessados em um processo de negócio específico.
   - Melhorar o tempo de resposta dos usuários finais ao reduzir o volume de dados acessados.
   - Suportar ferramentas de acesso como OLAP e data mining, que necessitam de estruturas de banco de dados internas específicas.
   - Simplificar o processo ETL (extração, transformação e carga) de dados, pois lidam com menos dados em comparação com um Data Warehouse completo.

- Reduzir custos de implementação em termos de tempo, dinheiro e recursos, comparado com um Data Warehouse completo.
- Definir claramente os usuários potenciais e obter suporte mais facilmente em projetos específicos de Data Mart em comparação com projetos de Data Warehouse.

Esses pontos destacam como os Data Marts são estruturados e implementados para fornecer análises específicas e ágeis para grupos de usuários dentro de uma organização, complementando o ambiente de Data Warehouse corporativo.

Metodologia

Projeto Conceitual de Banco de Dados Consiste em criar um modelo dos dados de uma organização,independente de detalhes físicos como SGBD, hardware ou desempenho.
Projeto Lógico de Banco de Dados Transforma o modelo conceitual em um modelo lógico, adequado para um tipo específico de modelo de dados, mas ainda sem considerar detalhes físicos específicos.
Projeto Físico de Banco de Dados Descreve a implementação do banco de dados em termos de armazenamento, organização de arquivos e índices, focando em eficiência de acesso e integridade, adaptado a um SGBD específico.

16.1.3 Fatores Críticos para o Sucesso no Design de Banco de Dados

Para garantir o sucesso no design de bancos de dados, as seguintes diretrizes são essenciais:

- **Interação com Usuários**: Trabalhar interativamente com os usuários tanto quanto possível.
- **Metodologia Estruturada**: Seguir uma metodologia estruturada durante o processo de modelagem de dados.
- **Abordagem Orientada a Dados**: Adotar uma abordagem centrada nos dados.
- **Considerações de Estrutura e Integridade**: Incorporar considerações estruturais e de integridade nos modelos de dados.
- **Combinação de Técnicas**: Integrar técnicas de conceitualização, normalização e validação de transações na metodologia de modelagem de dados.
- **Uso de Diagramas**: Utilizar diagramas para representar o máximo possível dos modelos de dados.
- **Uso de Linguagem de Design de Banco de Dados (DBDL)**: Usar DBDL para representar semânticas de dados adicionais que não podem ser facilmente representadas em diagramas.
- **Construção de Dicionário de Dados**: Criar um dicionário de dados para complementar os diagramas de modelo de dados e a DBDL.
- **Repetição de Passos**: Estar disposto a repetir os passos conforme necessário.

Esses fatores estão incorporados na metodologia apresentada para o design de banco de dados.

16.2 Visão Geral da Metodologia de Design de Banco de Dados

A metodologia de design de banco de dados pode ser dividida nas seguintes etapas:

Design Conceitual de Banco de Dados

Etapa 1: Construir o Modelo de Dados Conceitual

- 1.1 Identificar tipos de entidades
- 1.2 Identificar tipos de relacionamentos
- 1.3 Identificar e associar atributos a entidades ou relacionamentos
- 1.4 Determinar domínios de atributos
- 1.5 Determinar atributos candidatos, chaves primárias e alternadas
- 1.6 Considerar o uso de conceitos de modelagem avançada (opcional)
- 1.7 Verificar redundância no modelo
- 1.8 Validar o modelo conceitual de dados com as transações dos usuários
- 1.9 Revisar o modelo conceitual de dados com os usuários

Design Lógico de Banco de Dados para o Modelo Relacional

Etapa 2: Construir o Modelo de Dados Lógico

- **2.1 Derivar relações para o modelo de dados lógico**
- **2.2 Validar relações usando normalização**
- **2.3 Validar relações com as transações dos usuários**
- **2.4 Verificar restrições de integridade**
- **2.5 Revisar o modelo de dados lógico com os usuários**
- **2.6 Mesclar modelos de dados lógicos em um modelo global (opcional)**
- **2.7 Verificar crescimento futuro**

Design Físico de Banco de Dados para Bancos de Dados Relacionais

Etapa 3: Traduzir o Modelo de Dados Lógico para o SGBD Alvo

- **3.1 Projetar relações base**
- **3.2 Projetar representação de dados derivados**
- **3.3 Projetar restrições gerais**

Etapa 4: Projetar Organizações de Arquivos e Índices

- **4.1 Analisar transações**
- **4.2 Escolher organizações de arquivos**
- **4.3 Escolher índices**
- **4.4 Estimar requisitos de espaço em disco**

Etapa 5: Projetar Visões de Usuário

Etapa 6: Projetar Mecanismos de Segurança

Etapa 7: Considerar a Introdução de Redundância Controlada

Etapa 8: Monitorar e Otimizar o Sistema Operacional