semana 1- aula 01
Levantamento de requisitos
Introdução ao levantamento de requisitos
Código da aula: [SIS]ANO1C3B1S1A1

Objetivos da Aula:

- Conhecer os fundamentos sobre o levantamento de requisitos nos projetos de desenvolvimento de sistemas.
- Executar a manutenção de programas. Prestar apoio técnico na elaboração da documentação de sistemas.
- Conhecer frameworks de desenvolvimento ágeis, utilizando tecnologias de CI e CD que trabalham em conjunto com a segurança do ambiente funcional e as entregas divididas em partes que agregam valor ao negócio de forma rápida.
- Trabalhar a resolução de problemas no âmbito computacional.
- Recurso audiovisual para exibição de vídeos e imagens;
- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet

Exposição:

Slide 05

O levantamento de requisitos é a etapa inicial e crucial no desenvolvimento de software. É o processo de coletar, analisar e documentar as necessidades dos usuários e stakeholders para garantir que o software final atenda aos objetivos do negócio.

Em resumo, o levantamento de requisitos visa:

- Compreender as necessidades: Identificar o que o software precisa fazer para resolver problemas ou alcançar objetivos.
- Definir o escopo: Delimitar as funcionalidades e características do software.
- Garantir a qualidade: Estabelecer critérios para avaliar se o software atende às expectativas.
- Reduzir custos e tempo: Evitar retrabalho e garantir que o desenvolvimento siga o caminho certo.

Exemplo:

Imagine que uma empresa de e-commerce deseja criar um novo sistema de gerenciamento de pedidos. O levantamento de requisitos envolveria:

- Entrevistar os funcionários responsáveis pelo processamento de pedidos para entender suas necessidades e fluxos de trabalho.
- Analisar os dados de vendas e feedback dos clientes para identificar áreas de melhoria.
- Documentar os requisitos em um documento formal, incluindo:
    - Funcionalidades como rastreamento de pedidos, geração de faturas e gerenciamento de estoque.
    - Requisitos de desempenho, como tempo de resposta do sistema e capacidade de processar um grande volume de pedidos.
    - Requisitos de segurança, como proteção de dados do cliente e prevenção de fraudes.

Video:

- Para complementar sua compreensão sobre o tema, sugiro que assista ao seguinte vídeo:
    - Levantamento de Requisitos no desenvolvimento de software - Cedro Technologies
    - : “BÓSON TREINAMENTOS. O que é levantamento de requisitos - Tópicos de engenharia de software”. Disponível em:
    - https://www.youtube.com/watch?v=VcOeM2AD8Yk.

semana 1- aula 02
Levantamento de requisitos
Introdução ao levantamento de requisitos
Código da aula: [SIS]ANO1C3B1S1A2

Objetivos da Aula:

- ❖ Conhecer os fundamentos sobre o levantamento de requisitos nos projetos de desenvolvimento de sistemas.
- ❖ Executar a manutenção de programas. Prestar apoio técnico na elaboração da documentação de sistemas.
- ❖ Conhecer frameworks de desenvolvimento ágeis, utilizando tecnologias de CI e CD que trabalham em conjunto com a segurança do ambiente funcional e as entregas divididas em partes que agregam valor ao negócio de forma rápida.
- ❖ Trabalhar a resolução de problemas no âmbito computacional.
- ❖ Recurso audiovisual para exibição de vídeos e imagens;
- ❖ Caderno para anotações;
- ❖ Acesso ao laboratório de informática e/ou internet

Exposição:

Slide 8

As metodologias ágeis são um conjunto de práticas e princípios que visam tornar o desenvolvimento de software mais flexível, adaptável e focado na entrega de valor ao cliente. Elas surgiram como uma resposta às limitações das metodologias tradicionais, que muitas vezes resultam em projetos longos, caros e com pouca capacidade de adaptação às mudanças.

História do Manifesto Ágil:

Em fevereiro de 2001, 17 desenvolvedores de software se reuniram em Snowbird, Utah, para discutir alternativas às metodologias de desenvolvimento tradicionais. O resultado desse encontro foi o Manifesto Ágil, um documento que define os quatro valores e os doze princípios que norteiam as metodologias ágeis:

Valores do Manifesto Ágil:

- Indivíduos e interações, mais que processos e ferramentas.
- Software em funcionamento, mais que documentação abrangente.
- Colaboração com o cliente, mais que negociação de contratos.
- Responder a mudanças, mais que seguir um plano.[1]

Principais Métodos e Conceitos:

- Scrum: Um framework iterativo e incremental que divide o trabalho em ciclos curtos chamados sprints.
- Kanban: Um método que utiliza um quadro visual para gerenciar o fluxo de trabalho e limitar o trabalho em progresso.
- XP (Extreme Programming): Uma metodologia que enfatiza práticas como testes automatizados, integração contínua e programação em pares.
- Lean: Uma abordagem que busca eliminar desperdícios e otimizar o fluxo de valor.

Principais Conceitos:

- Iterações: O trabalho é dividido em ciclos curtos e repetitivos, permitindo feedback constante e adaptação.
- Incrementos: O software é desenvolvido e entregue em partes funcionais, permitindo que o cliente comece a utilizá-lo o mais cedo possível.
- Feedback: A comunicação constante com o cliente e a equipe é essencial para garantir que o software atenda às necessidades do usuário.
- Adaptação: A capacidade de responder a mudanças e ajustar o plano de acordo com as necessidades do projeto.

Diferença entre Metodologias Ágeis e Tradicionais:

Metodologias Ágeis	Metodologias Tradicionais
Flexibilidade e adaptabilidade	Rigidez e controle
Foco na entrega de valor ao cliente	Foco no cumprimento do plano
Iterações curtas e feedback constante	Etapas longas e sequenciais
Colaboração e comunicação constante	Documentação extensa e comunicação formal
Mudanças são bem-vindas	Mudanças são vistas como obstáculos

Em resumo, as metodologias ágeis representam uma mudança de paradigma no desenvolvimento de software, priorizando a colaboração, a flexibilidade e a entrega de valor ao cliente.

video :“ATTEKITA DEV. Esqueça isso e seu projeto estará condenado (Análise de Requisitos)”. Disponível em: https://www.youtube.com/watch?v=rVbJ7ykuLig.

semana 1- aula 03
Levantamento de requisitos
Introdução ao levantamento de requisitos
Código da aula: [SIS]ANO1C3B1S1A3

Objetivos da Aula:

- Conhecer os fundamentos sobre o levantamento de requisitos nos projetos de desenvolvimento de sistemas.
- Executar a manutenção de programas. Prestar apoio técnico na elaboração da documentação de sistemas.
- Conhecer frameworks de desenvolvimento ágeis, utilizando tecnologias de CI e CD que trabalham em conjunto com a segurança do ambiente funcional e

as entregas divididas em partes que agregam valor ao negócio de forma rápida.

- Trabalhar a resolução de problemas no âmbito computacional.
- Recurso audiovisual para exibição de vídeos e imagens;
- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet

Exposição:

Slide 8 a 11:

vídeo da aula: “ATTEKITA DEV. Prototipagem, saiba porque essa etapa é fundamental”. “ATTEKITA DEV. Prototipagem, saiba porque essa etapa é fundamental”. Disponível em: https://www.youtube.com/watch?v=8YbAHNCv9-w.

semana 2- aula 01
Testes funcionais e não funcionais
Introdução aos testes de software e tipos de testes
Código da aula: [SIS]ANO1C3B1S2A1

Objetivos da Aula:

- Conhecer os conceitos acerca de testes de software, princípios e práticas essenciais. • Desenvolver soluções de software (back-end, front-end e full-stack) utilizando técnicas, métodos, ferramentas e linguagens de programação diversas;
- Executar manutenção de programas;
- Resolução de problemas.
- Recurso audiovisual para exibição de vídeos e imagens;
- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet

Exposição:

Slide 7 até 09:

Testes de software são um conjunto de atividades realizadas para verificar se um software funciona conforme o esperado e atende aos requisitos estabelecidos. Eles são essenciais para garantir a qualidade, confiabilidade e segurança do software.

Objetivos dos testes de software:

- Identificar e corrigir defeitos: Os testes ajudam a encontrar bugs, falhas e inconsistências no software, permitindo que sejam corrigidos antes que o software seja lançado para o público.
- Validar requisitos: Os testes verificam se o software atende aos requisitos funcionais e não funcionais especificados pelos usuários e stakeholders.
- Melhorar a qualidade: Os testes ajudam a identificar áreas de melhoria no software, como desempenho, usabilidade e segurança.
- Reduzir riscos: Os testes ajudam a reduzir o risco de falhas no software, que podem ter consequências graves para a empresa e seus clientes.

Exemplos de tipos de testes de software:

- Teste unitário: Testa unidades individuais de código, como funções ou métodos, para verificar se estão funcionando corretamente.
- Teste de integração: Testa a interação entre diferentes componentes do software para garantir que eles funcionem juntos como esperado.
- Teste de sistema: Testa o sistema como um todo para verificar se atende aos requisitos e funciona corretamente em diferentes ambientes.
- Teste de aceitação: Testa o software do ponto de vista do usuário final para verificar se atende às suas expectativas e necessidades.
- Teste de regressão: Testa o software após alterações para garantir que as novas funcionalidades não afetem a funcionalidade existente.

Recomendações de vídeos:

- Entenda os 7 princípios do Teste de Software que todo engenheiro de software deve saber
- Testes de Software | Teste Caixa-Branca e Teste Caixa-Preta - Teste Estrutural e Teste Funcional
- PLANEJANDO TESTES DE SOFTWARE DO ZERO
- TESTE DE SOFTWARE: por onde começar
- Níveis e Técnicas de Teste, Entenda! unidade, integração, sistema, regressão, funcional, estrutural

vídeo: PESSONIZANDO. TESTE DE SOFTWARE: por onde começar. Disponível em: https://www.youtube.com/watch?v=NnamjfPYuiY. Acesso em: 4 jan. 2024.

semana 2- aula 02
Testes funcionais e não funcionais
Introdução aos testes de software e tipos de testes
Código da aula: [SIS]ANO1C3B1S2A2

Objetivos da Aula:

- ❖ Conhecer os conceitos acerca de testes de software, princípios e práticas essenciais. • Desenvolver soluções de software (back-end, front-end e full-stack) utilizando técnicas, métodos, ferramentas e linguagens de programação diversas;
- ❖ Executar manutenção de programas;
- ❖ Resolução de problemas.
- ❖ Recurso audiovisual para exibição de vídeos e imagens;
- ❖ Caderno para anotações;
- ❖ Acesso ao laboratório de informática e/ou internet

Exposição:

Slide 8/9:

Os testes de regressão são um tipo de teste de software realizado após realizar alterações no código, como correções de bugs, novas funcionalidades ou otimizações. O objetivo principal é garantir que as modificações não introduziram novos defeitos ou causaram efeitos colaterais indesejados em partes já existentes e funcionais do software.

Em outras palavras, os testes de regressão servem para verificar se o software "regrediu" em termos de qualidade e funcionalidade após as alterações. É uma medida preventiva para assegurar a estabilidade do sistema.

Objetivo principal dos testes de regressão:

- Verificar se as alterações não quebraram funcionalidades existentes: Garantir que as partes do software que já estavam funcionando continuam funcionando corretamente após as modificações.
- Identificar efeitos colaterais inesperados: Descobrir se as alterações em uma área do código impactaram outras áreas do sistema de forma negativa.
- Manter a qualidade do software ao longo do tempo: Assegurar que o software permanece estável e confiável à medida que evolui.

Exemplos de cenários onde testes de regressão são necessários:

- Correção de um bug: Após corrigir um erro específico, os testes de regressão garantem que a correção não introduziu novos problemas em outras partes do sistema.
- Implementação de uma nova funcionalidade: Ao adicionar uma nova funcionalidade, os testes de regressão verificam se essa nova parte interage corretamente com as funcionalidades existentes e não as prejudica.

- Otimização de performance: Após otimizar o código para melhorar o desempenho, os testes de regressão garantem que a otimização não afetou a funcionalidade do software.
- Alterações na infraestrutura: Mudanças no ambiente de hospedagem ou em bibliotecas externas podem exigir testes de regressão para garantir a compatibilidade e o funcionamento correto do software.

Exemplos de ferramentas mais utilizadas para testes de regressão:

Existem diversas ferramentas que auxiliam na execução e gerenciamento de testes de regressão, muitas vezes com funcionalidades de automação para facilitar o processo. Algumas das mais populares incluem:

- Selenium: Uma ferramenta de automação de testes de código aberto para aplicações web. Permite escrever scripts em diversas linguagens de programação para simular a interação de um usuário com o navegador.
- JUnit e TestNG: Frameworks de teste para a linguagem Java, amplamente utilizados para testes unitários e de integração, que também podem ser aplicados em testes de regressão.
- NUnit: Um framework de teste para a linguagem .NET, similar ao JUnit/TestNG, utilizado para testes unitários e de integração em aplicações .NET.
- Cypress: Um framework de teste de ponta a ponta (end-to-end) para aplicações web, focado em tornar os testes mais rápidos, fáceis e confiáveis.
- Playwright: Outro framework poderoso para automação de testes de ponta a ponta, desenvolvido pela Microsoft, que suporta diversos navegadores e linguagens.
- Appium: Uma ferramenta de automação para testar aplicações móveis (Android e iOS).
- Jenkins: Uma ferramenta de integração contínua e entrega contínua (CI/CD) que pode ser integrada com outras ferramentas de teste para automatizar a execução dos testes de regressão a cada nova alteração no código.
- qTest: Uma plataforma de gerenciamento de testes que oferece recursos para planejar, executar e rastrear testes, incluindo testes de regressão.

A escolha da ferramenta depende das necessidades específicas do projeto, da tecnologia utilizada e da expertise da equipe de testes. A automação é frequentemente utilizada em testes de regressão devido à necessidade de executar os mesmos testes repetidamente após cada alteração.

video
CÓDIGO FONTE TV. O QUE FAZ UM ANALISTA DE TESTES?. Disponível em: https://www.youtube.com/watch?v=O7rB5XoakEc. Acesso em: 4 jan. 2024

semana 2- aula 03
Testes funcionais e não funcionais
Introdução aos testes de software e tipos de testes
Código da aula: [SIS]ANO1C3B1S2A3

Objetivos da Aula:

- Conhecer os conceitos acerca de testes de software, princípios e práticas essenciais. • Desenvolver soluções de software (back-end, front-end e full-stack) utilizando técnicas, métodos, ferramentas e linguagens de programação diversas;
- Executar manutenção de programas;
- Resolução de problemas.
- Recurso audiovisual para exibição de vídeos e imagens;
- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet

Exposição:

Slide 5

Etapas para Desenvolver um Plano de Testes

Um plano de testes é um documento essencial que descreve a abordagem, os recursos e o cronograma das atividades de teste para um projeto de software. Seguir etapas estruturadas garante que o plano seja abrangente e eficaz. Aqui estão as etapas principais:

1. Definição do Escopo e Objetivos:
    - Identificar o que será testado: Delimitar as funcionalidades, módulos e áreas do sistema que serão cobertas pelos testes.
    - Definir os objetivos dos testes: Estabelecer o que se espera alcançar com os testes (ex: garantir a funcionalidade, identificar defeitos críticos, validar requisitos).
    - Identificar os critérios de aceitação: Definir os critérios que o software deve atender para ser considerado aprovado nos testes.
2. Análise dos Requisitos:
    - Revisar a documentação de requisitos: Compreender completamente as necessidades do cliente e as especificações do sistema.
    - Identificar os requisitos testáveis: Garantir que os requisitos sejam claros, concisos e possam ser verificados através de testes.

    - Priorizar os requisitos: Identificar os requisitos mais críticos para focar os esforços de teste.
3. Estratégia de Testes:

    - Definir os tipos de testes a serem realizados: Escolher os tipos de testes adequados (ex: funcional, não funcional, usabilidade, segurança, desempenho).
    - Definir as técnicas de teste a serem utilizadas: Escolher as técnicas específicas para cada tipo de teste (ex: teste de caixa preta, teste de caixa branca, teste exploratório).
    - Definir o nível de cobertura dos testes: Estabelecer a profundidade dos testes (ex: testes de caminho básico, testes de limite).
4. Recursos:

    - Identificar a equipe de testes: Definir os papéis e responsabilidades dos membros da equipe.
    - Identificar o ambiente de testes: Especificar o hardware, software e dados necessários para realizar os testes.
    - Identificar as ferramentas de teste: Listar as ferramentas que serão utilizadas para automação, gerenciamento de testes, etc.
5. Cronograma e Estimativa:

    - Estimar o esforço de teste: Calcular o tempo e os recursos necessários para cada atividade de teste.
    - Definir o cronograma de testes: Estabelecer as datas de início e fim para as diferentes fases de teste.
    - Identificar os marcos de teste: Definir os pontos de verificação importantes durante o processo de teste.
6. Gerenciamento de Riscos:

    - Identificar os riscos potenciais: Listar os fatores que podem impactar o sucesso dos testes (ex: atrasos, falta de recursos).
    - Avaliar a probabilidade e o impacto dos riscos: Analisar a chance de ocorrência de cada risco e suas possíveis consequências.
    - Definir planos de mitigação: Estabelecer ações para prevenir ou minimizar os impactos dos riscos identificados.
7. Métricas e Relatórios:

    - Definir as métricas de teste: Escolher as métricas que serão utilizadas para acompanhar o progresso e a qualidade dos testes (ex: número de casos de teste executados, número de defeitos encontrados).
    - Definir os tipos de relatórios: Especificar os relatórios que serão gerados para comunicar o status dos testes (ex: relatório de progresso, relatório de defeitos).

8. Aprovação e Controle de Versão:

    - Obter a aprovação do plano de testes: Garantir que as partes interessadas revisem e aprovem o plano.
    - Estabelecer um processo de controle de versão: Gerenciar as diferentes versões do plano de testes à medida que ele é atualizado.

Exemplo de Plano de Testes (Simplificado)

Projeto: Sistema de E-commerce

1. Escopo e Objetivos:

- Escopo: Testar as funcionalidades de cadastro de usuário, visualização de produtos, carrinho de compras e checkout.
- Objetivos: Garantir que as funcionalidades principais operem corretamente, identificar defeitos críticos e validar os requisitos funcionais.
- Critérios de Aceitação: Todas as funcionalidades testadas devem operar sem erros críticos, e os principais fluxos de usuário devem ser concluídos com sucesso.

2. Análise de Requisitos:

- Os requisitos funcionais foram documentados em um documento de especificação de requisitos (SRS).
- Os requisitos testáveis incluem:
    - Cadastro de usuário com validação de campos.
    - Visualização de detalhes do produto.
    - Adição e remoção de itens do carrinho.
    - Processo de checkout com cálculo de frete e impostos.
- Prioridade alta para os requisitos relacionados ao carrinho de compras e checkout.

3. Estratégia de Testes:

- Tipos de Testes: Testes funcionais, testes de usabilidade (básicos).
- Técnicas de Teste: Teste de caixa preta (análise de valor limite, particionamento de equivalência), teste exploratório.
- Cobertura: Teste de caminho básico para os fluxos principais.

4. Recursos:

- Equipe de Testes: 2 testadores.
- Ambiente de Testes: Navegadores Chrome e Firefox, banco de dados de teste.

- Ferramentas de Teste: Selenium WebDriver (para automação básica), Jira (para gerenciamento de defeitos).

5. Cronograma e Estimativa:

- Esforço de Teste Estimado: 5 dias úteis.
- Cronograma:
    - Dia 1-2: Preparação do ambiente e casos de teste.
    - Dia 3-4: Execução dos testes.
    - Dia 5: Relatório de testes e retestes.
- Marcos: Conclusão da criação dos casos de teste, conclusão da primeira rodada de testes.

6. Gerenciamento de Riscos:

- Riscos: Atraso na entrega do build, indisponibilidade do ambiente de testes.
- Probabilidade/Impacto: Médio/Médio para ambos.
- Mitigação: Comunicação constante com a equipe de desenvolvimento, provisionamento antecipado do ambiente.

7. Métricas e Relatórios:

- Métricas: Número de casos de teste executados, número de defeitos encontrados, número de defeitos corrigidos.
- Relatórios: Relatório de progresso diário, relatório final de testes com resumo dos defeitos.

8. Aprovação e Controle de Versão:

- Aprovação do plano de testes por: Gerente de Projeto, Líder de Desenvolvimento.
- Controle de Versão: Utilizar sistema de controle de versão (ex: Git) para gerenciar as atualizações do plano.

Este é um exemplo simplificado. Um plano de testes real seria muito mais detalhado e específico para o projeto em questão. No entanto, ele ilustra as principais seções e informações que devem ser incluídas.

Slide 8

Vídeo

CÓDIGO FONTE TV. Como ser um Analista QA de Sucesso! Disponível em: https://www.youtube.com/watch?v=eCmIALYTqPk. Acesso em: 4 jan. 2024.

semana 3- aula 01
Tipos de modelagem de dados
Introdução à modelagem de dados e conceitos fundamentais
Código da aula: [SIS]ANO1C3B1S3A1

Objetivos da Aula:

- Os conceitos sobre a modelagem de dados no contexto de desenvolvimento de software e sua importância.
- Desenvolver soluções de software (back-end, front-end e full-stack) utilizando técnicas, métodos, ferramentas e linguagens de programação diversas;
- Executar manutenção de programas;
- Resolver problemas computacionais
- Recurso audiovisual para exibição de vídeos e imagens;
- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet

Exposição:

Slide 5

A modelagem de dados é o processo de criar uma representação conceitual da informação necessária para um sistema de software, um banco de dados ou um processo de negócios. É como desenhar um mapa que mostra quais dados são importantes, como eles se relacionam entre si e quais regras governam esses dados.

O objetivo principal da modelagem de dados é organizar a informação de forma eficiente, precisa e compreensível, facilitando o desenvolvimento de sistemas que armazenam, recuperam e manipulam esses dados de maneira eficaz.

Em termos práticos, a modelagem de dados envolve:

1. Identificar as entidades: As entidades são os objetos ou conceitos do mundo real que precisam ser representados no sistema. Pense em substantivos.
2. Definir os atributos: Os atributos são as características ou propriedades das entidades. Pense nos adjetivos que descrevem as entidades.
3. Estabelecer os relacionamentos: Os relacionamentos descrevem como as entidades se conectam umas às outras. Pense em verbos que conectam as entidades.
4. Especificar as regras de negócio: As regras de negócio são as restrições e condições que governam os dados.

Exemplos Práticos de Modelagem de Dados:

Exemplo 1: Sistema de Biblioteca

- Entidades:
  - Livro
  - Autor
  - Usuário
  - Empréstimo
- Atributos:
  - Livro: Título, ISBN, Ano de Publicação, Número de Páginas
  - Autor: Nome, Sobrenome, Nacionalidade
  - Usuário: ID, Nome, Endereço, Telefone
  - Empréstimo: Data de Empréstimo, Data de Devolução
- Relacionamentos:
  - Um Livro pode ter um ou mais Autores (relacionamento muitos-para-muitos).
  - Um Usuário pode pegar emprestado muitos Livros (relacionamento muitos-para-muitos).
  - Um Empréstimo envolve um Usuário e um Livro (relacionamento um-para-um com cada).
- Regras de Negócio:
  - Um livro não pode ser emprestado se não houver cópias disponíveis.
  - Um usuário não pode ter mais de 5 livros emprestados ao mesmo tempo.
  - A data de devolução não pode ser anterior à data de empréstimo.

Exemplo 2: Sistema de E-commerce

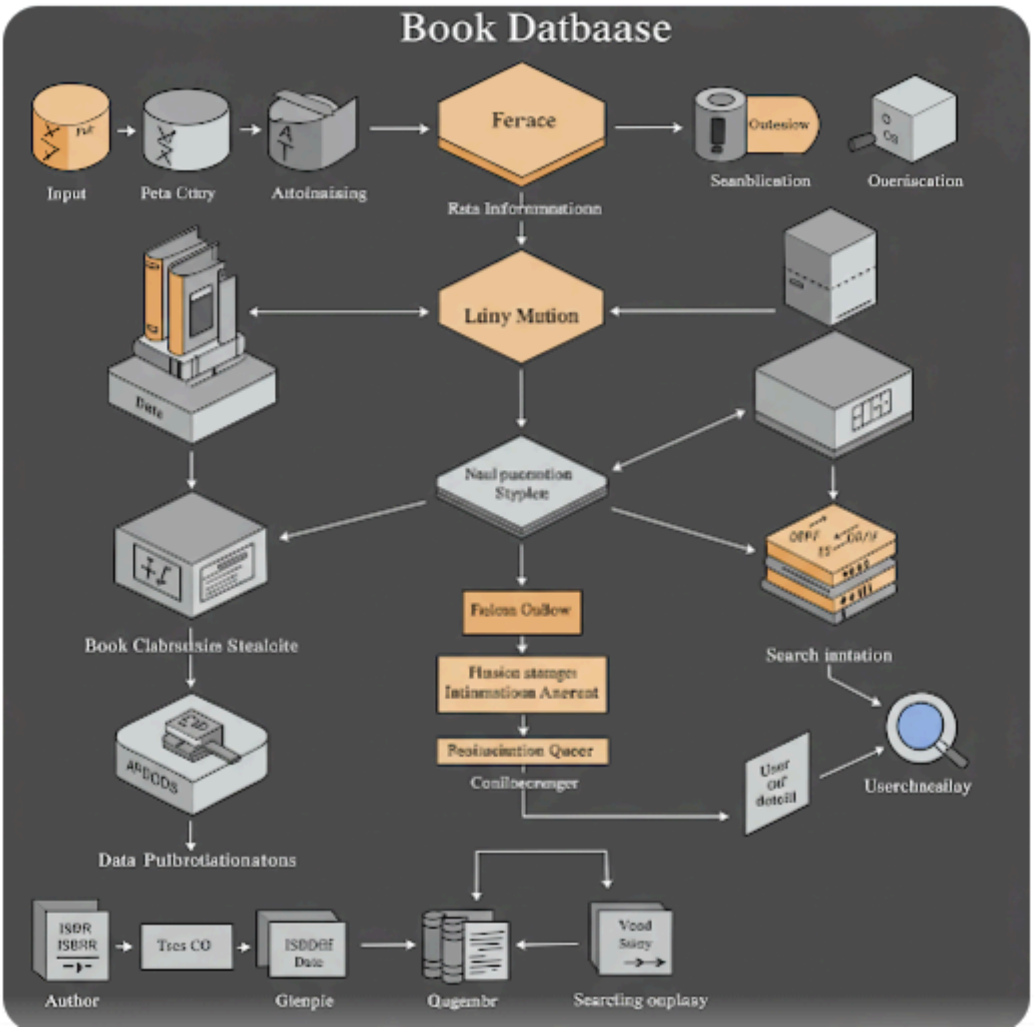
- Entidades:
  - Cliente
  - Produto
  - Pedido
  - Item do Pedido
- Atributos:
  - Cliente: ID, Nome, Email, Endereço
  - Produto: ID, Nome, Descrição, Preço, Estoque
  - Pedido: Número do Pedido, Data do Pedido, Status do Pedido, ID do Cliente
  - Item do Pedido: Quantidade, Preço Unitário, ID do Pedido, ID do Produto
- Relacionamentos:
  - Um Cliente pode fazer muitos Pedidos (relacionamento um-para-muitos).
  - Um Pedido contém muitos Itens do Pedido (relacionamento um-para-muitos).

    - Um `Item do Pedido` se refere a um `Produto` (relacionamento muitos-para-um).
- Regras de Negócio:
    - O estoque de um produto deve ser atualizado após a conclusão de um pedido.
    - O preço total do pedido é a soma do preço unitário multiplicado pela quantidade de cada item.
    - Um pedido só pode ser enviado se todos os itens estiverem em estoque.

Exemplo 3: Sistema de Gerenciamento de Projetos

- Entidades:
    - `Projeto`
    - `Tarefa`
    - `Membro da Equipe`
    - `Atribuição`
- Atributos:
    - `Projeto`: `Nome`, `Data de Início`, `Data de Fim`, `Descrição`
    - `Tarefa`: `Nome`, `Descrição`, `Data de Início`, `Data de Fim`, `Status`, `Prioridade`
    - `Membro da Equipe`: `ID`, `Nome`, `Email`, `Função`
    - `Atribuição`: `Data de Início`, `Data de Fim`, `ID da Tarefa`, `ID do Membro da Equipe`
- Relacionamentos:
    - Um `Projeto` pode ter muitas `Tarefas` (relacionamento um-para-muitos).
    - Uma `Tarefa` pode ser atribuída a um ou mais `Membros da Equipe` (relacionamento muitos-para-muitos).
- Regras de Negócio:
    - A data de fim de uma tarefa não pode ser anterior à data de início.
    - Um membro da equipe só pode ser atribuído a tarefas dentro do projeto em que está envolvido.

Abaixo a imagem do que seria este relacionamento:



Em resumo, a modelagem de dados é fundamental para:

- Organizar e estruturar a informação: Facilitando o armazenamento e a recuperação eficiente dos dados.
- Garantir a consistência e a integridade dos dados: Definindo regras e relacionamentos que evitam inconsistências e erros.
- Facilitar a comunicação entre desenvolvedores, analistas e usuários: Fornecendo um modelo visual e compreensível dos dados.
- Servir como base para o desenvolvimento de bancos de dados e aplicações: Guiando a criação da estrutura física dos dados.

Ao criar um modelo de dados bem definido, é possível construir sistemas mais robustos, eficientes e que atendam às necessidades dos usuários de forma eficaz.

Slide 12

A normalização de dados é um processo fundamental no design de bancos de dados relacionais. Seu objetivo principal é organizar os dados de forma eficiente

para minimizar a redundância e a dependência, garantindo assim a integridade e a consistência dos dados.

Em termos mais simples, a normalização busca estruturar as tabelas do banco de dados de maneira que cada informação seja armazenada apenas uma vez e que as tabelas estejam relacionadas de forma lógica. Isso evita problemas como:

- Redundância de dados: A mesma informação sendo repetida em várias linhas.
- Anomalias de inserção: Dificuldade em inserir novos dados sem informações relacionadas.
- Anomalias de exclusão: Ao excluir uma informação, perde-se outras informações relacionadas.
- Anomalias de atualização: Alterar uma informação requer a atualização em várias linhas.

A normalização é alcançada através da aplicação de um conjunto de regras chamadas Formas Normais (FN). As mais comuns são a Primeira Forma Normal (1FN), a Segunda Forma Normal (2FN) e a Terceira Forma Normal (3FN). Existem formas mais elevadas (4FN, 5FN, etc.), mas geralmente a 3FN é suficiente para a maioria das aplicações.

Conceitos Chave:

- Chave Primária: Um ou mais atributos que identificam exclusivamente cada linha em uma tabela.
- Chave Estrangeira: Um atributo (ou conjunto de atributos) em uma tabela que referencia a chave primária de outra tabela. É usada para estabelecer relacionamentos entre tabelas.
- Dependência Funcional: Um atributo (ou conjunto de atributos) determina univocamente outro atributo. Por exemplo, em uma tabela de clientes, o CPF determina o nome do cliente.

Exemplos Práticos de Normalização:

Exemplo 1: Tabela Não Normalizada de Pedidos

Imagine uma tabela de pedidos com a seguinte estrutura:

ID do Pedido	Nome do Cliente	Endereço do Cliente	Produto	Quantidade	Preço Unitário

1	Ana Silva	Rua A, 123	Camiseta	2	25.00
1	Ana Silva	Rua A, 123	Calça	1	50.00
2	João Pereira	Rua B, 456	Tênis	1	100.00
2	João Pereira	Rua B, 456	Meias	3	10.00

Problemas:

- Redundância: O nome e o endereço do cliente são repetidos para cada item do pedido.
- Anomalia de atualização: Se o endereço da Ana mudar, é preciso atualizar em duas linhas.
- Anomalia de inserção: Não é possível inserir um novo cliente sem um pedido.

Normalização para a 1FN:

A 1FN exige que cada coluna contenha apenas valores atômicos (indivisíveis) e que não haja grupos repetidos de colunas. A tabela acima já está na 1FN.

Normalização para a 2FN:

A 2FN exige que a tabela esteja na 1FN e que todos os atributos não chave dependam totalmente da chave primária. A chave primária da tabela acima é a combinação de `ID do Pedido` e `Produto`. O nome e o endereço do cliente dependem apenas do `ID do Pedido`, não do `Produto`.

Solução para a 2FN: Criar duas tabelas:

Tabela Pedidos:

ID do Pedido (PK)	Nome do Cliente	Endereço do Cliente
1	Ana Silva	Rua A, 123

2	João Pereira	Rua B, 456

Tabela Itens do Pedido:

ID do Pedido (FK)	Produto (PK)	Quantidade	Preço Unitário
1	Camiseta	2	25.00
1	Calça	1	50.00
2	Tênis	1	100.00
2	Meias	3	10.00

Agora, a informação do cliente está armazenada apenas na tabela `Pedidos`, eliminando a redundância.

Normalização para a 3FN:

A 3FN exige que a tabela esteja na 2FN e que nenhum atributo não chave dependa de outro atributo não chave. No exemplo acima, ambas as tabelas já estão na 3FN.

Exemplo 2: Tabela de Funcionários com Informações do Departamento

Imagine uma tabela de funcionários:

ID do Funcionário	Nome do Funcionário	ID do Departamento	Nome do Departamento	Localização do Departamento
1	Maria Oliveira	101	Vendas	São Paulo

2	Pedro Almeida	102	Marketing	Rio de Janeiro
3	Ana Souza	101	Vendas	São Paulo

Problemas:

- Redundância: O nome e a localização do departamento são repetidos para cada funcionário do mesmo departamento.
- Anomalia de atualização: Se a localização do departamento de Vendas mudar, é preciso atualizar em várias linhas.
- Anomalia de inserção: Não é possível inserir um novo departamento sem um funcionário.

Normalização para a 3FN:

Criar duas tabelas:

Tabela Funcionários:

ID do Funcionário (PK)	Nome do Funcionário	ID do Departamento (FK)
1	Maria Oliveira	101
2	Pedro Almeida	102
3	Ana Souza	101

Tabela Departamentos:

ID do Departamento (PK)	Nome do Departamento	Localização do Departamento
101	Vendas	São Paulo

102	Marketing	Rio de Janeiro

Novamente, a redundância foi eliminada e a integridade dos dados foi melhorada.

Benefícios da Normalização:

- Redução da redundância de dados: Economiza espaço de armazenamento e facilita a manutenção.
- Melhora da integridade dos dados: Evita inconsistências e erros.
- Facilita a atualização e a manutenção do banco de dados: Alterações em uma informação precisam ser feitas em apenas um lugar.
- Torna o banco de dados mais flexível: Facilita a criação de novas consultas e relatórios.

Em resumo, a normalização de dados é um processo essencial para projetar bancos de dados relacionais eficientes e confiáveis, garantindo a qualidade e a consistência da informação armazenada.

vídeo “8 etapas para Modelagem de Dados”, do canal HASHTAG PROGRAMAÇÃO . Disponível em: https://www.youtube.com/watch?v=UZca ZD5VG0g. Acesso em: 11 jan. 2024.

semana 3- aula 02
Tipos de modelagem de dados
Introdução à modelagem de dados e conceitos fundamentais
Código da aula: [SIS]ANO1C3B1S3A2

Objetivos da Aula:

- ❖ Os conceitos sobre a modelagem de dados no contexto de desenvolvimento de software e sua importância.
- ❖ Desenvolver soluções de software (back-end, front-end e full-stack) utilizando técnicas, métodos, ferramentas e linguagens de programação diversas;
- ❖ Executar manutenção de programas;
- ❖ Resolver problemas computacionais
- ❖ Recurso audiovisual para exibição de vídeos e imagens;
- ❖ Caderno para anotações;
- ❖ Acesso ao laboratório de informática e/ou internet

Exposição:

Slide 10/11

Em modelagem de dados, a integridade dos dados refere-se à precisão, consistência e confiabilidade dos dados armazenados no banco de dados ao longo do tempo. É a garantia de que os dados estão corretos, completos e em conformidade com as regras e restrições definidas no modelo.

Em resumo, integridade de dados significa que os dados são o que deveriam ser e permanecem assim.

Exemplos Práticos de Integridade de Dados:

1. Restrição de Chave Primária:
    - Conceito: Garante que cada linha em uma tabela seja única.
    - Exemplo: Em uma tabela de `Clientes`, o `ID do Cliente` é definido como chave primária. Isso impede a inserção de dois clientes com o mesmo ID. Se alguém tentar cadastrar um cliente com um ID já existente, o sistema rejeitará a operação, mantendo a integridade da unicidade dos clientes.
2. Restrição de Chave Estrangeira:
    - Conceito: Garante que o valor em uma coluna de uma tabela (chave estrangeira) corresponda a um valor existente na chave primária de outra tabela relacionada.
    - Exemplo: Em uma tabela de `Pedidos`, a coluna `ID do Cliente` é uma chave estrangeira que referencia a chave primária `ID do Cliente` na tabela `Clientes`. Isso garante que só seja possível associar um pedido a um cliente que realmente existe no sistema. Se um pedido for criado com um `ID do Cliente` inexistente, o sistema impedirá a operação, mantendo a integridade do relacionamento entre pedidos e clientes.
3. Restrição de Valor Não Nulo (NOT NULL):
    - Conceito: Garante que uma determinada coluna não possa conter valores vazios.
    - Exemplo: Na tabela de `Produtos`, a coluna `Nome do Produto` é definida como `NOT NULL`. Isso garante que todos os produtos cadastrados tenham um nome. Se alguém tentar cadastrar um produto sem informar o nome, o sistema impedirá a operação, garantindo a integridade da informação básica do produto.
4. Restrição de Domínio (CHECK):
    - Conceito: Define um conjunto de valores válidos que podem ser inseridos em uma coluna.

    - Exemplo: Na tabela de `Pedidos`, a coluna `Status do Pedido` pode ter uma restrição `CHECK` que permite apenas os valores 'Pendente', 'Em Processamento', 'Enviado' e 'Entregue'. Se alguém tentar inserir um status diferente, o sistema rejeitará a operação, mantendo a integridade dos possíveis estados de um pedido.
5. Restrição de Unicidade (UNIQUE):

    - Conceito: Garante que os valores em uma coluna (ou combinação de colunas) sejam únicos em toda a tabela, permitindo valores nulos (diferente da chave primária).
    - Exemplo: Na tabela de `Clientes`, a coluna `Email` pode ter uma restrição `UNIQUE`. Isso garante que não haja dois clientes com o mesmo endereço de e-mail.

Em resumo, a integridade dos dados é crucial porque:

- Garante a qualidade da informação: Permite tomar decisões corretas baseadas em dados confiáveis.
- Evita erros e inconsistências: Reduz a chance de problemas e falhas no sistema.
- Facilita a análise e a geração de relatórios: Dados precisos levam a insights mais confiáveis.
- Assegura a conformidade com regulamentações: Em muitos setores, a integridade dos dados é uma exigência legal.

Ao implementar corretamente as restrições de integridade durante a modelagem de dados, é possível construir sistemas mais robustos e confiáveis, onde a informação é precisa e consistente.

semana 3- aula 02
Tipos de modelagem de dados
Introdução à modelagem de dados e conceitos fundamentais
Código da aula: [SIS]ANO1C3B1S3A3

Objetivos da Aula:

- ❖ Os conceitos sobre a modelagem de dados no contexto de desenvolvimento de software e sua importância.
- ❖ Desenvolver soluções de software (back-end, front-end e full-stack) utilizando técnicas, métodos, ferramentas e linguagens de programação diversas;
- ❖ Executar manutenção de programas;

- ❖ Resolver problemas computacionais
- ❖ Recurso audiovisual para exibição de vídeos e imagens;
- ❖ Caderno para anotações;
- ❖ Acesso ao laboratório de informática e/ou internet

Exposição:

Slide 14

Para ilustrar a arquitetura de dados na prática, aqui estão exemplos de tecnologias comumente usadas em diferentes estágios do ciclo de vida dos dados:

1. Coleta e Ingestão de Dados:

- Sistemas de Mensageria:
  - Apache Kafka: Plataforma de streaming de eventos distribuída, altamente escalável e tolerante a falhas, ideal para coletar grandes volumes de dados em tempo real de diversas fontes (aplicações, sensores, etc.).
  - RabbitMQ: Broker de mensagens robusto e flexível, usado para comunicação assíncrona entre sistemas e coleta de dados.
- Ferramentas ETL (Extract, Transform, Load) / ELT (Extract, Load, Transform):
  - Apache NiFi: Plataforma visual para automatizar o fluxo de dados entre sistemas, facilitando a coleta e o roteamento de dados.
  - Informatica PowerCenter: Ferramenta comercial robusta para integração e transformação de dados.
  - AWS Glue: Serviço serverless da AWS para ETL, permitindo transformar e mover dados entre data stores.
  - Apache Airflow: Plataforma para orquestração de workflows de dados, incluindo tarefas de coleta e processamento.
- APIs (Application Programming Interfaces):
  - Tecnologias para construir APIs como REST (usando frameworks como Flask em Python, Spring Boot em Java, Node.js com Express) permitem que diferentes sistemas troquem dados de forma estruturada.
- Webhooks: Mecanismo para que uma aplicação envie dados para outra em tempo real quando um evento específico acontece.

2. Armazenamento de Dados:

- Bancos de Dados Relacionais (SQL):
  - PostgreSQL: Sistema de gerenciamento de banco de dados relacional de código aberto, robusto e extensível.

    - MySQL: Banco de dados relacional de código aberto, amplamente utilizado para aplicações web.
    - SQL Server: Banco de dados relacional desenvolvido pela Microsoft.
    - Oracle Database: Sistema de gerenciamento de banco de dados relacional comercial, conhecido por sua robustez e escalabilidade.
- Bancos de Dados NoSQL:
    - MongoDB: Banco de dados orientado a documentos, ideal para dados não estruturados ou semiestruturados.
    - Cassandra: Banco de dados NoSQL distribuído, projetado para alta disponibilidade e escalabilidade horizontal, adequado para grandes volumes de dados.
    - Redis: Armazenamento de dados em memória, usado como cache, broker de mensagens e banco de dados NoSQL.
    - Amazon DynamoDB: Serviço de banco de dados NoSQL totalmente gerenciado e altamente escalável da AWS.
- Data Warehouses:
    - Amazon Redshift: Serviço de data warehouse rápido e escalável da AWS, otimizado para consultas analíticas.
    - Snowflake: Plataforma de data warehouse baseada na nuvem, conhecida por sua escalabilidade e facilidade de uso.
    - Google BigQuery: Serviço de data warehouse serverless e altamente escalável do Google Cloud.
    - Apache Hive: Sistema de data warehouse construído sobre o Hadoop, permitindo consultas SQL sobre grandes conjuntos de dados.
- Data Lakes:
    - Apache Hadoop: Framework para armazenamento e processamento distribuído de grandes conjuntos de dados.
    - Amazon S3 (Simple Storage Service): Serviço de armazenamento de objetos altamente escalável e durável da AWS, frequentemente usado como base para data lakes.
    - Azure Data Lake Storage: Serviço de armazenamento escalável para big data no Azure.
    - Google Cloud Storage: Serviço de armazenamento de objetos escalável e durável do Google Cloud.

3. Processamento e Transformação de Dados:

- Frameworks de Processamento Distribuído:
    - Apache Spark: Motor de processamento de dados rápido e versátil, capaz de lidar com processamento em batch e streaming.
    - Apache Flink: Framework de processamento de streaming de dados em tempo real.

    - Apache Beam: Modelo de programação unificado para processamento de dados em lote e streaming, que pode ser executado em diferentes engines (Spark, Flink, Google Cloud Dataflow).
- Linguagens de Programação:
    - Python: Amplamente utilizado para análise de dados, machine learning e scripts de processamento de dados com bibliotecas como Pandas, NumPy e Scikit-learn.
    - SQL: Linguagem padrão para consultar e manipular dados em bancos de dados relacionais e data warehouses.
    - Java e Scala: Linguagens frequentemente usadas com frameworks como Spark e Flink para processamento de big data.

4. Análise e Visualização de Dados:

- Ferramentas de Business Intelligence (BI) e Visualização:
    - Tableau: Plataforma de análise visual de dados interativa.
    - Power BI: Ferramenta de análise de negócios da Microsoft.
    - Looker: Plataforma de BI e análise de dados baseada na nuvem (agora parte do Google Cloud).
    - Qlik Sense: Plataforma de análise de dados que permite explorar dados de forma associativa.
    - Apache Superset: Plataforma de exploração e visualização de dados moderna e de código aberto.
- Linguagens de Programação e Bibliotecas:
    - Python: Com bibliotecas como Matplotlib, Seaborn e Plotly para criar visualizações personalizadas.
    - R: Linguagem de programação estatística e para visualização de dados.
- Plataformas de Ciência de Dados e Machine Learning:
    - Jupyter Notebooks/JupyterLab: Ambientes interativos para escrever e compartilhar código, visualizações e texto explicativo.
    - TensorFlow e PyTorch: Frameworks de código aberto para machine learning e deep learning.
    - Databricks: Plataforma unificada para engenharia de dados, ciência de dados e machine learning, baseada no Apache Spark.
    - AWS SageMaker, Azure Machine Learning, Google AI Platform: Plataformas de machine learning gerenciadas na nuvem.

5. Governança e Segurança de Dados:

- Ferramentas de Catálogo de Dados:
    - Apache Atlas: Framework de governança de dados e metadados de código aberto para o Hadoop.

    - AWS Glue Data Catalog: Serviço de catálogo de metadados para dados no AWS Glue e Amazon S3.
    - Azure Purview: Serviço de governança de dados unificado no Azure.
    - Alation Data Catalog: Plataforma comercial para catalogação e governança de dados.
- Ferramentas de Qualidade de Dados:
    - Trifacta: Plataforma para preparação e limpeza de dados.
    - Informatica Data Quality: Ferramenta comercial para garantir a qualidade dos dados.
- Sistemas de Gerenciamento de Segurança:
    - Tecnologias de criptografia, controle de acesso (RBAC, ABAC), mascaramento de dados e auditoria.
    - Serviços de segurança oferecidos por provedores de nuvem (AWS IAM, Azure Active Directory, Google Cloud IAM).

Importância da Arquitetura de Dados na Escolha de Tecnologias:

A arquitetura de dados define os requisitos e as diretrizes para a seleção e a implementação dessas tecnologias. Por exemplo:

- Se a necessidade é processar grandes volumes de dados em tempo real, tecnologias como Apache Kafka e Apache Flink podem ser escolhidas.
- Se a prioridade é a análise de dados estruturados para BI, um data warehouse como Snowflake ou Amazon Redshift pode ser mais adequado.
- Se a empresa lida com muitos dados não estruturados e precisa de flexibilidade, um data lake baseado em Amazon S3 ou Azure Data Lake Storage pode ser a melhor opção, com um banco de dados NoSQL como MongoDB para acesso rápido.

É importante notar que a arquitetura de dados não se limita a tecnologias específicas, mas sim à forma como essas tecnologias são organizadas e interagem para atender aos objetivos de dados da organização. A escolha das tecnologias deve ser guiada pelos requisitos da arquitetura e pelas necessidades do negócio.