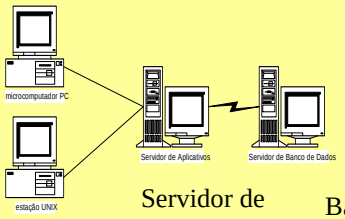
Arquitetura WEB para client/server de três camadas



Browser

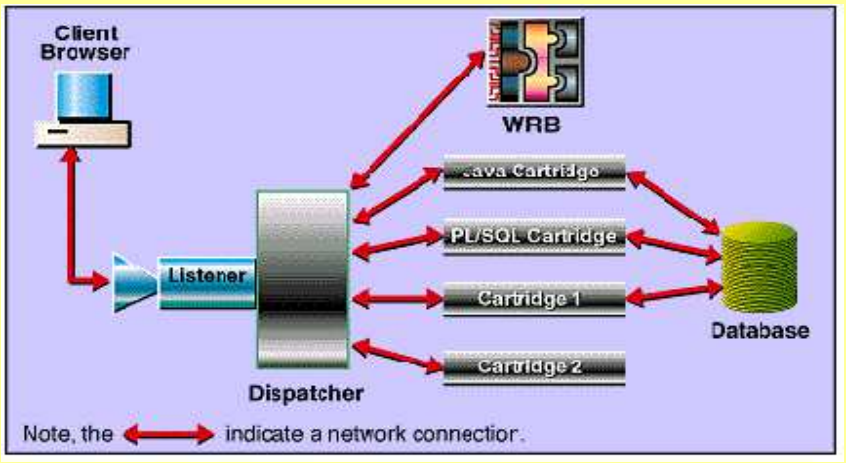
Servidor de WEB

Banco de Dados

Essa arquitetura considera os clientes como qualquer máquina que tenha um browser capaz de entender formulários e “HTML”.

1

Estrutura do Oracle Web Server



2

Listener - Verifica o tipo de página requisitada pelo browser.

Dispatcher- Através da URL verifica se o pedido é para uma página estática ou para um cartridge. Se for uma página estática, envia a mesma para o cliente, se for um cartridge envia o pedido para o respectivo cartridge.

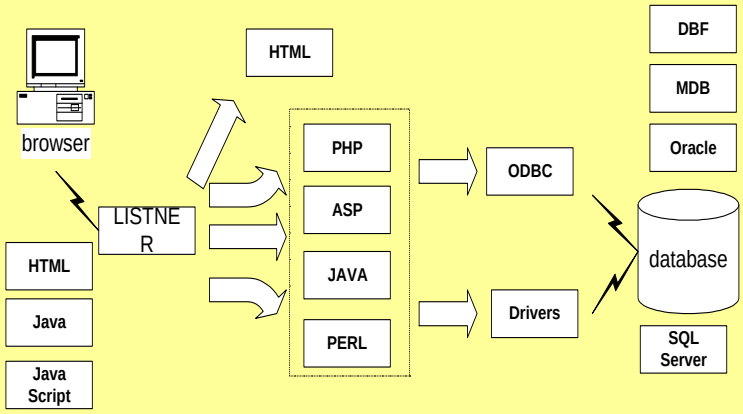
WRB - Organiza os pedidos de clientes aos cartridges e cria os processos referentes a cada um desses pedidos.

Cartridges - Responsável pela conecção com o servidor de dados, e passagem de parâmetros, obtendo o resultado e retornando-o para o cliente.

Database - Base de dados, responável pelo armazenamento das informações.

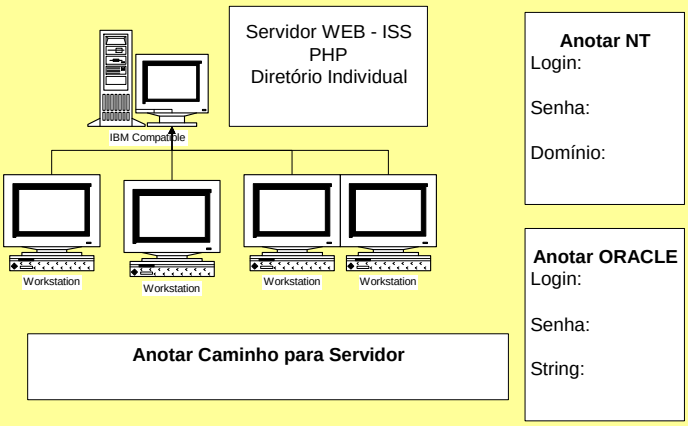
3

Acesso a Base de Dados



4

Estrutura do Curso



5

Base de Dados

- É um conjunto de dados que representam uma entidade do mundo real. Conjunto de dados próprios à um domínio de aplicação (Biblioteca)
- Uma Base de Dados representa o conjunto (coerente, integrado e partilhado) das informações necessárias ao funcionamento de uma empresa.
- O gerenciamento é assegurado por um software chamada Sistema de Gerenciamento de Banco de dados (SGBD).
- Conjunto de Dados Integrados que tem por objetivo atender a uma comunidade de usuários.

6

Processo de Confecção de Banco de Dados

- Modelo Conceitual
- Modelo Lógico
- Modelo Físico

- Manipulação de Dados através de uma linguagem de manipulação de dados.

7

Modelos de Dados

- Modelos de Dados
  - Descreve os dados
    - Objeto Ligação Propriedade
  - Descreve as regras de Restrição
- Esquemas
  - Expressão da descrição dos dados

8

Modelo Conceitual

- Esquema Conceitual das Necessidades
  - Descrição primária da realidade a ser implementada.
  - Descrição das regras que regem o negócio.
    - Cada livro terá um código de eletrônico.
    - Os empréstimos poderão ser realizados para 14 dias...
  - Rascunho do empreendimento a ser modelado.

9

Modelo Lógico

- Esquema Conceitual
  - No modelo utilizado pelo Equipamento
  - Dependente do SGBD
    - Tabela de Livros
      - Código do Livro,
      - Autores
      - Título
      - Ano
      - Editora
  - Fase de Implementação

10

Modelo Físico

- Esquema Interno
  - Como os dados são armazenados no computador.
  - Arquivo seqüencial, utilizando linguagem C.
  - Busca binárias, em árvore

11

Estrutura de Informações

- Esquema Externo
  - Conjunto de Informações para cada grupo de usuários.
    - Alunos e professores
      - Lista de livros
      - Consulta ao acervo
    - Bibliotecárias
      - Lista de livros emprestados
      - Usuários em débito com a biblioteca
  - Fase de utilização

12

SGBD

- Sistema Gerenciador de Banco de Dados
  - Redundância de Dados
    - Dados repetidos
  - Inconsistência de Dados
    - Livro consta como disponível mas esta emprestado a um usuário.
  - Acesso Simultâneo às Informações
    - Dois usuários emprestando o mesmo exemplar de livro.
  - Controle de Segurança

13

Arquitetura de um SGBD

Interface Usuário

Interface de Acesso Físco

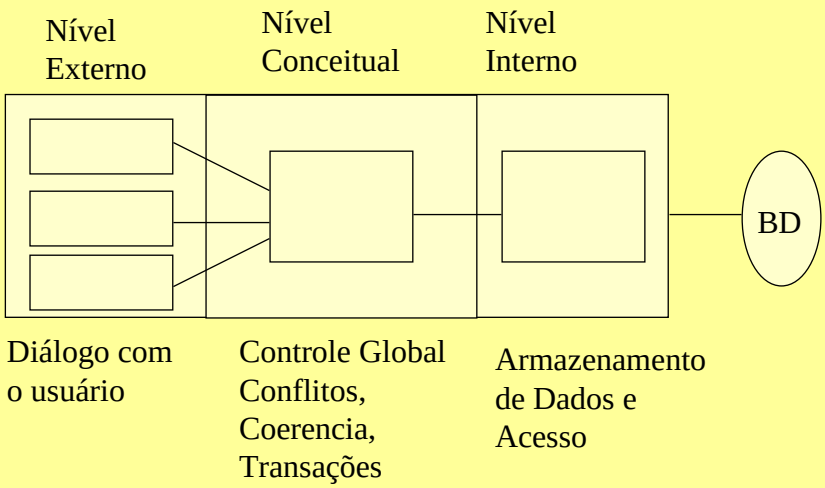
14

Arquitetura de um SGBD

- Interface Usuário
  - permite aos usuários a expressão de seus requerimentos
    - Definição : DDL (Data Definition Language)
    - Interrogação e Modificação: DML (Data Manipulation Language)
    - Interfaces de com outros produtos.
  - Compreensão, Análise e Verificação das Perguntas
  - Interface agradável
  - Linguagens de mais alto nível possível
- Interface de Acesso Físico
  - otimização do armazenamento dos dados
  - pouco espaço em disco
  - alta velocidade de acesso

15

Arquitetura Interna -SGBD



16

SGBD Funcionamento

- Nível Externo
  - O usuário exprime uma pergunta
  - Análise Sintática: linguagem válida
  - Análise Semântica: objetos no esquema externo
  - Reorganiza e mostra aos usuários
- Nível Conceitual
  - Correspondência objetos EE -> EC
  - Controle de proteção
- Nível Interno
  - Controle de concorrência
  - Tradução em primitivos de acesso físico

17

Modelos de Dados

- Entidade-Relacionamento (conceitual)
- Relacional
- Extensões (aplicações não convencionais)
  - ER e Relacional Estendidos
  - Orientado à Objetos

FORMAL	VALORES LEGAIS
Relação	Tabela
Tupla	Registro
Atributo	Campo
Chave Primária	Identificador Único
Domínio	Grupo de Valores Legais

18

Domínio

- É a menor unidade de dados no modelo relacional.
- É um valor de dado individual
- É um conjunto de tais valores.
- Todos do mesmo tipo.
- Devem ser Atômicos (valores não decomponíveis).
  - Código do Aluno: [0 a 999999]
  - Idade: [0 a 150]
  - Estado: Todos estados da União
- Domínio são grupos de valores, a partir dos quais os valores reais aparecem especificados como ATRIBUTOS.
- Se dois atributos retiram seus valores do mesmo domínio as comparações farão sentido.

19

Domínio

Select nome, endereco from alunos
where endereco in
(select sigla from estadosdauniao) ERRADO

Select nome, endereco from alunos
where estado in
(select sigla from estadosdauniao) CORRETO

20

Modelo Conceitual

- Obter uma descrição abstrata, independente de implementação em computador, dos dados que serão armazenados no banco de dados.
- O modelo Entidade-Relacionamento é a técnica de modelagem mais difundida.
- Também chamado Modelo ER.
- Representação gráfica através do Diagrama Entidade Relacionamento.
- Criada 1976 por Peter Chen.

21

Conceitos

- A idéia fundamental deste modelo é a de conservar como conceitos de base os conceitos genéricos (objetos, associação, propriedade) usados no processo de abstração que vai da observação de uma realidade à sua descrição.

Entidade	Objetos
Associação	Ligação entre os objetos
Atributo	Propriedade dos objetos.

22

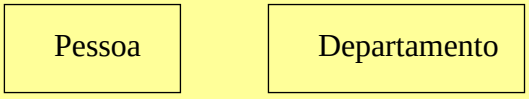
Entidades

- É um conjunto de objetos da realidade modelada sobre os quais deseja-se manter informações no banco de dados.
- Um entidade é o objeto do mundo real (concreto ou abstrato) que deseja-se representar no computador e que tem uma existência própria.
- Uma entidade existe independentemente do fato que ela está ligada à outros objetos do BD.
- Pode representar objetos concretos (pessoa, automóvel) como objetos abstratos (departamento, histórico)

23

Representação Gráfica

- Entidade é representada por um retângulo.
- Cada retângulo representa um conjunto de objetos sobre os quais deseja-se guardar informações.

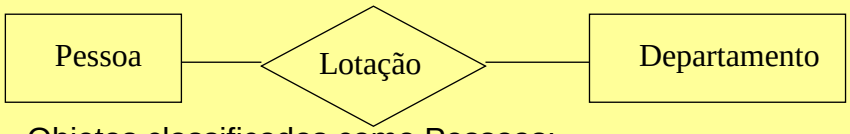


- Para se referir a um objeto particular fala-se em ocorrência de entidade ou instância de entidade.
  - Pessoa Caio Nakashima

24

Relacionamento

- Conjunto de associações entre entidades.
- Sua representação é através de um losango, ligado por linhas aos retângulos (Entidades)

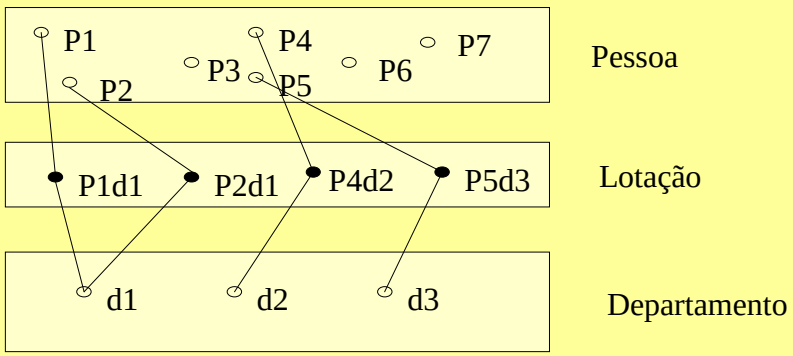


- Objetos classificados como Pessoas;
- Objetos classificados como Departamentos;
- Associações que ligam Pessoas aos Departamentos (Lotação)

25

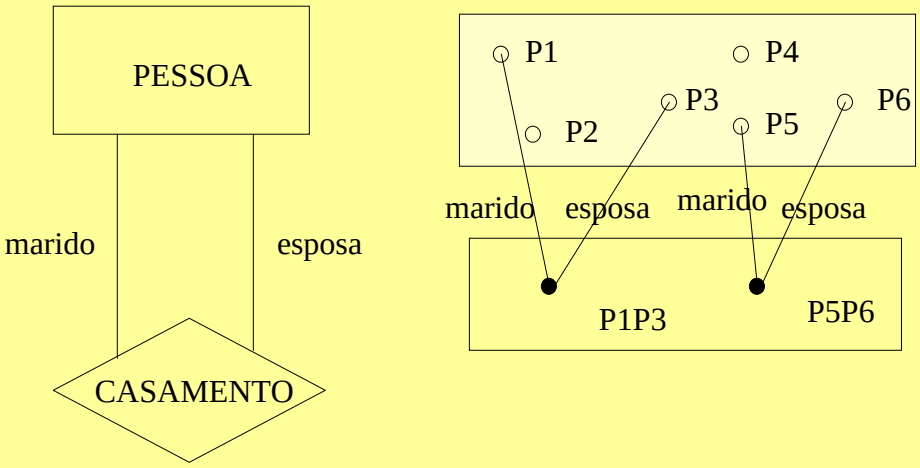
Relacionamento

- Ocorrência de um relacionamento
  - O Caio Nakashima esta lotado no DAINF
- Diagrama de Ocorrências



26

Exemplo de Auto Relacionamento

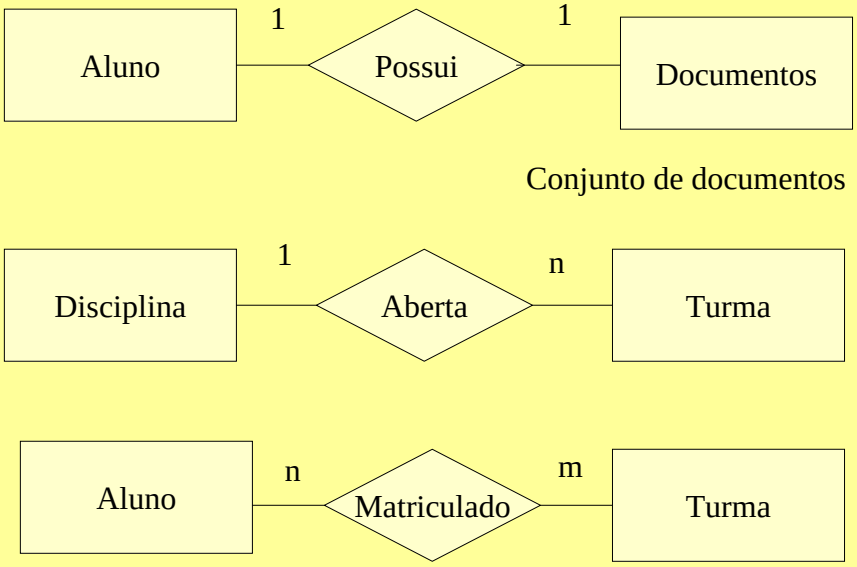


27

Cardinalidade de Relacionamentos

- Número mínimo/máximo de ocorrências de entidade associadas a uma ocorrência da entidade em questão através do relacionamento.

28

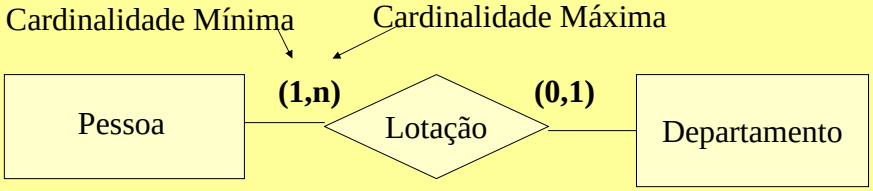


Classificação de Relacionamentos

29

Cardinalidade Mínima

- É o número mínimo de ocorrências de entidade que são associadas a uma ocorrência de uma entidade através de um relacionamento.
- Cardinalidade mínima 0 (associação opcional)
- Cardinalidade mínima 1 (associação obrigatória)



30

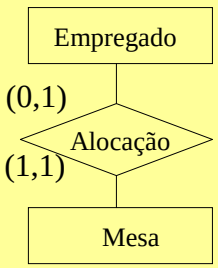
Cardinalidade Mínima

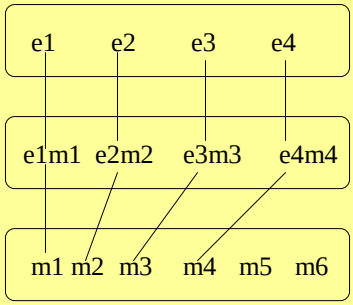
- É o número mínimo de ocorrências de entidade que são associadas a uma ocorrência de uma entidade através de um relacionamento.
- Cardinalidade Mínima (0)
  - Associação Opcional
- Cardinalidade Mínima (1)
  - Associação Obrigatória
  - Indica que o relacionamento deve obrigatoriamente associar uma ocorrência de entidade a cada ocorrência da entidade em questão

31

Exemplo de Cardinalidade Mínima

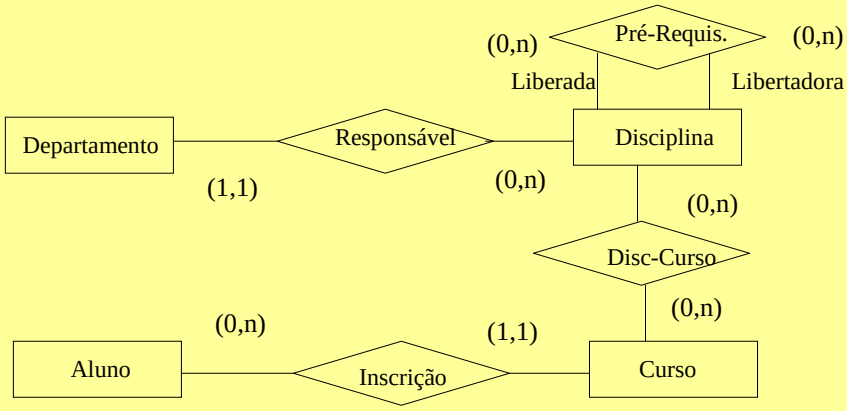
- A cardinalidade mínima é usada para especificar que cada empregado deve ter a ele alocada obrigatoriamente uma mesa (cardinalidade mínima 1) e que uma mesa pode existir sem que a ela esteja alocado um empregado (cardinalidade mínima 0).





32

Exemplo de Entidade e Relacionamento



Deseja-se manter informações sobre alunos, cursos, disciplinas e departamentos.

33

Exemplos de Cardinalidade

- Cada disciplina possui exatamente uma departamento responsável, e um departamento é responsável por muitas disciplinas, inclusive por nenhuma.
- Uma disciplina pode possuir diversos pré-requisitos, inclusive nenhum. Uma disciplina pode ter pré-requisito de muitas outras disciplinas, inclusive nenhuma.

34

Exemplo de Cardinalidade (2)

- Uma disciplina pode aparecer no currículo de muitos cursos (inclusive nenhum) e um curso pode possuir muitas disciplinas em seu currículo (inclusive nenhuma).
- Um aluno esta inscrito em exatamente um curso e um curso pode ter nele inscritos muitos alunos (inclusive nenhum).

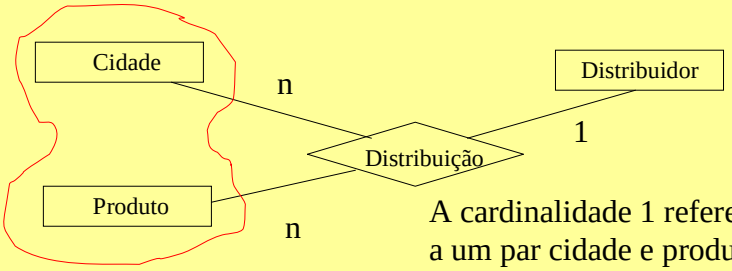
35

Relacionamento Ternário

- Pode-se definir relacionamentos de grau maior do que dois (relacionamentos ternários, quaternários, etc.)
- A cardinalidade refere-se a pares de entidades.
- Em um relacionamento R entre três entidades A,B e C a cardinalidade máxima de A e B dentro de R indica quantas ocorrências de C podem estar associadas a um par de ocorrências de A e B.

36

Exemplo



A cardinalidade 1 refere-se a um par cidade e produto

37

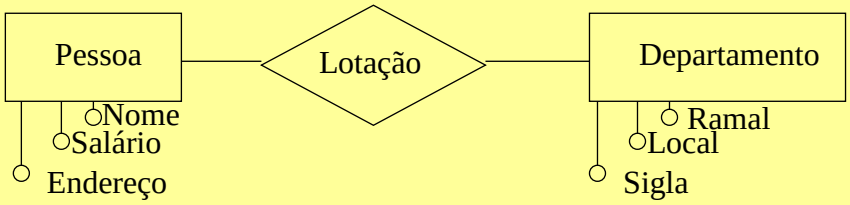
Exemplo (cont.)

- O 1 na linha que liga o retângulo representativo da entidade Distribuidor ao losango representativo do relacionamento expressa que cada par de ocorrências (cidade, produto) esta associado a no máximo um distribuidor. (Não há concorrência pela distribuição de um produto na mesma cidade)
- Cidade e distribuidor podem estar associados muitos produtos. Um distribuidor pode distribuir em uma cidade muitos produtos.
- Produto e distribuidor podem estar associadas muitas cidades, ou em outros termos um distribuidor pode distribuir um produto em muitas cidades.

38

Atributo

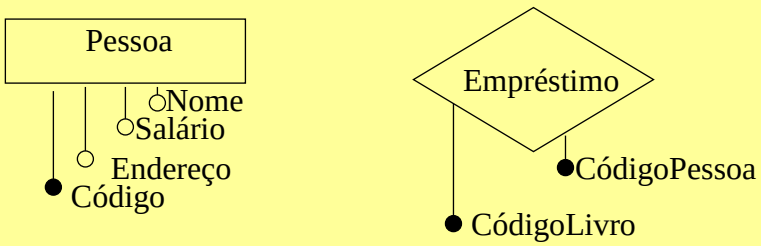
- Dado que é associado a cada ocorrência de uma entidade ou de um relacionamento.
- Um atributo é a propriedade associada à uma Entidade, ou à um Relacionamento
- Exemplos:
  - nome, salário, endereço (para a Entidade Pessoa)
  - sigla, nome, local, ramal, chefe (para a entidade Departamento)



39

Identificando Entidades

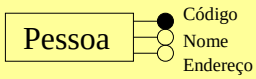
- Cada entidade deve possuir um identificador.
- Identificador é um conjunto de um ou mais atributos que identificam uma única instância do registro.
- Os identificadores são representados por um círculo preto.



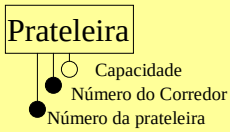
40

Identificador de Entidades

- É um conjunto de um ou mais atributos cujos valores servem para distinguir uma ocorrência da entidade das demais ocorrências da mesma entidade.
- No DER, os atributos identificadores são representados por um círculo preto.



- Uma entidade também pode ser identificado por mais de um atributo.



41

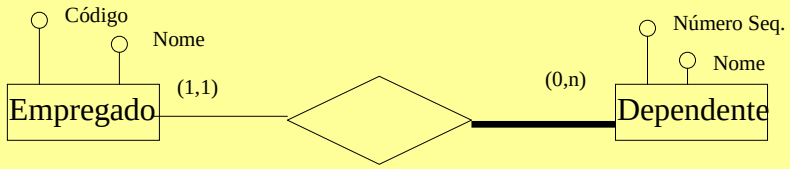
Propriedades

- O identificador é mínimo
  - Isto significa que o identificador de uma entidade deve ser composto de tal forma que, retirando um dos atributos ou relacionamentos que o compõe, ela deixa de ser um identificador.
- Cada entidade deve possuir um único identificador.
  - Em alguns casos, diferentes conjuntos de atributos podem servir para distinguir as ocorrências de entidade.
  - Empregado (Código ou CPF)
  - A escolha do identificador de entidade é arbitrada pelo projetista do sistema.
- Não deve ser nulo
- Não deve possuir repetição deste valor.

42

Relacionamento Identificador

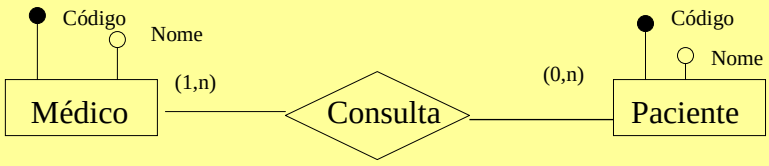
- É o identificador composto não somente por atributos da própria entidade mas também por relacionamentos dos quais a entidade participa.
- Esta entidade também é denominada fraca.
- Sua existência esta condicionada a outra entidade e usa o relacionamento como identificador.
- No DER, o relacionamento usado como identificador é indicado por uma linha mais densa.



43

Identificando Relacionamentos

- Os relacionamentos também devem ser identificados em alguns casos.
  - Um relacionamento é identificado pelas entidades de participantes, bem como pelos atributos identificadores.



44

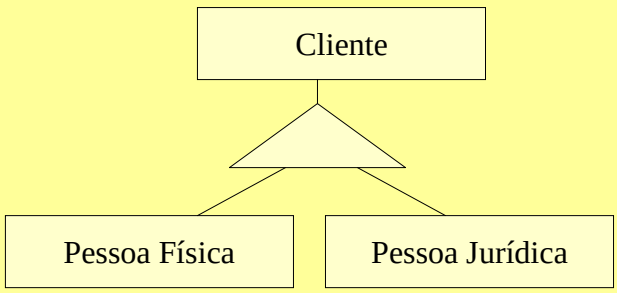
Exercício

- Apresente ao menos cinco exemplos dos conceitos básico da abordagem ER apresentados: entidade, relacionamento, atributo.
- Explique a diferença entre uma entidade e uma ocorrência de entidade. Explique.
- Dê um exemplo de um relacionamento ternário. Escolha e justifique as cardinalidade mínima e máxima.
- Apresente três exemplos de entidades com relacionamentos identificadores (entidades fracas)

45

Generalização/Especialização

- Propriedades podem ser atribuídas a entidades através do conceito de generalização /especialização.
- Pode-se atribuir propriedades particulares a um subconjunto das ocorrências (especializadas) de um entidade genérica.



46

Propriedades

- Uma entidade pode ser especializada em qualquer número de entidades, inclusive uma única.
  - Se apenas os motoristas possuíssem propriedades particulares, haveria apenas uma entidade especializada (motoristas).
- Não há limite no número de níveis hierárquicos da generalização / especialização.
  - Uma entidade especializada em uma generalização / especialização pode ser entidade genérica em uma outra generalização / especialização.

47

Exemplo

- A entidade Pessoa Física possui, além de seus atributos particulares, CIC e Sexo, também todas as propriedades da ocorrência CLIENTE correspondente, ou seja, os atributos Nome e Código.
- Seu identificador (Código) e o relacionamento com a entidade Filial.
- Toda pessoa jurídica tem como atributos nome, código, CGC e tipo de organização, é identificada pelo código e esta obrigatoriamente relacionada a exatamente uma Filial.

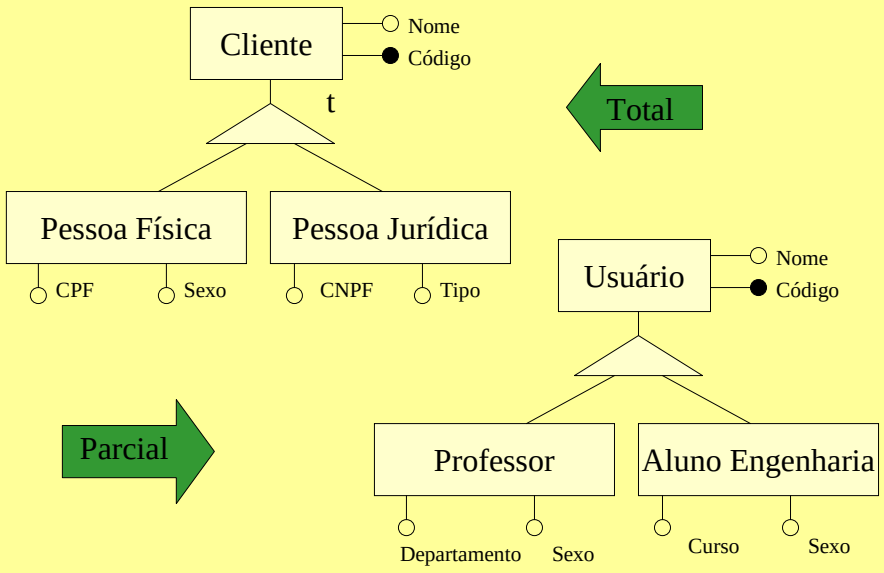
48

Classificação

- Total (t)
  - Para cada ocorrência da entidade genérica existe sempre uma ocorrência em uma das entidades especializadas.
  - Em toda ocorrência da entidade Cliente corresponde uma ocorrência em uma das duas especializações.
- Parcial (p)
  - Nem toda ocorrência da entidade genérica possui uma ocorrência correspondente em uma entidade especializada.
  - Nem todo usuário da Biblioteca do CEFET-PR é professor ou aluno da Engenharia.

49

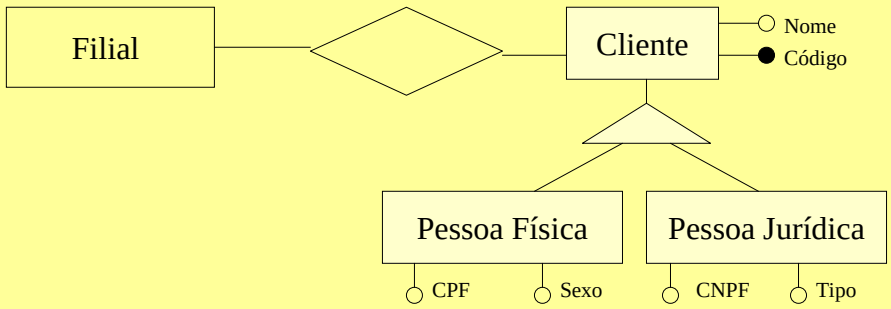
Representação Gráfica



50

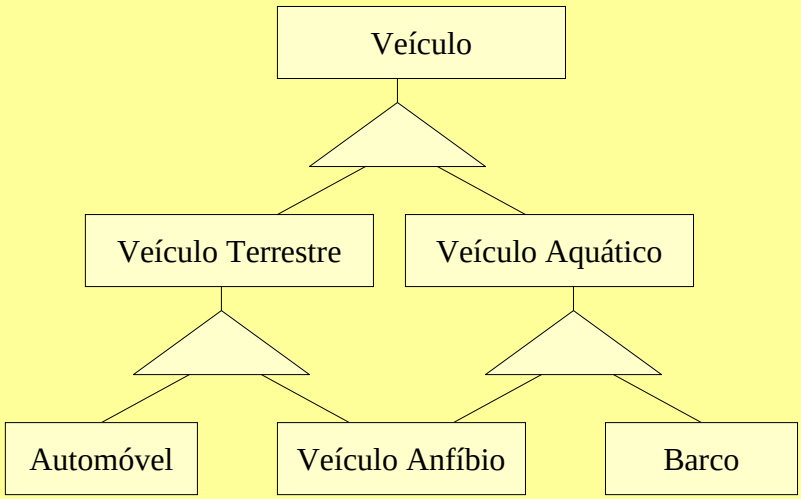
Herança de Propriedades

- Cada ocorrência da entidade especializada possui, além de suas próprias propriedades (atributos, relacionamentos e generalizações / especializações), também as propriedades da ocorrência da entidade genérica correspondente.



51

Exemplo



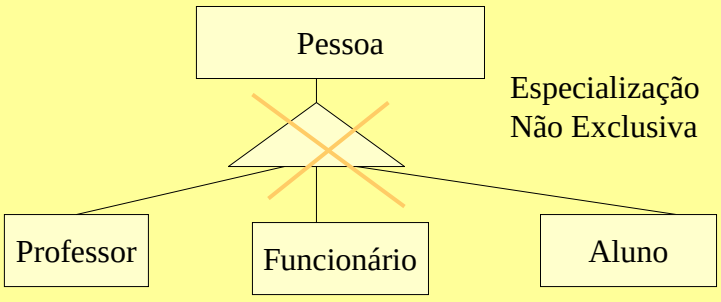
52

Tipos

- Especialização Exclusiva
  - Significa que uma ocorrência de entidade genérica aparece, para cada hierarquia generalização / especialização no máximo uma vez.
  - Cada veículo pode ser um automóvel ou veículo anfíbio ou barco e somente um destes.
- Especialização Generalização Não Exclusiva
  - Significa que uma ocorrência de entidade genérica aparece, para cada hierarquia generalização / especialização várias vezes.
  - Não podem herdar o identificador da entidade genérica.

53

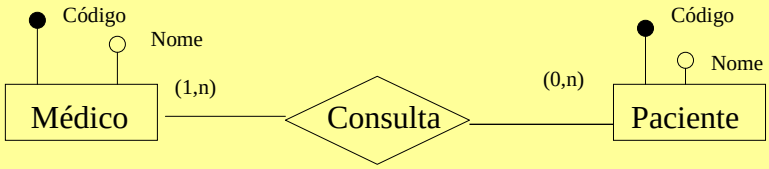
Exemplo



54

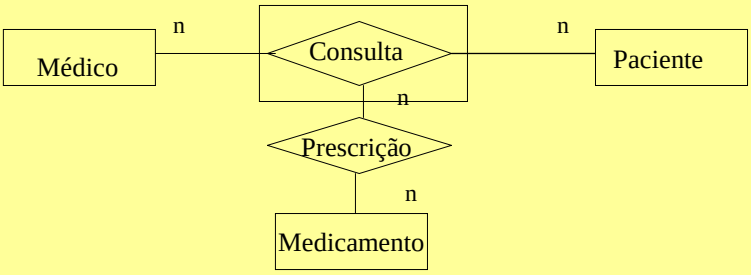
Entidade Associativa

- Um relacionamento é uma associação entre entidades.
- Em ER não foi prevista a possibilidade de associar uma entidade com um relacionamento ou então associar dois relacionamentos entre si.
- Exemplo. Dado a situação abaixo:
  - Registrar a informação de que em cada consulta um ou mais medicamentos podem ser prescritos ao paciente.



55

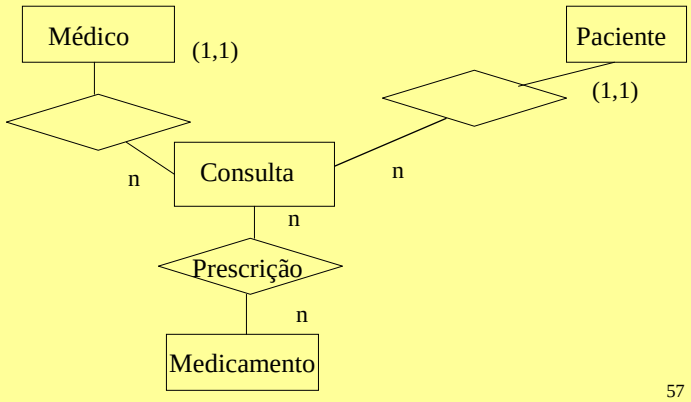
Exemplo



- A entidade Medicamento deve ser ligado à consulta, pois se for ligado ao médico, perde-se a informação sobre o paciente e vice-versa.
- Deseja-se relacionar o medicamento à consulta.
- Uma entidade associativa é a redefinição de um relacionamento, que passa a ser tratado como se fosse uma entidade.

56

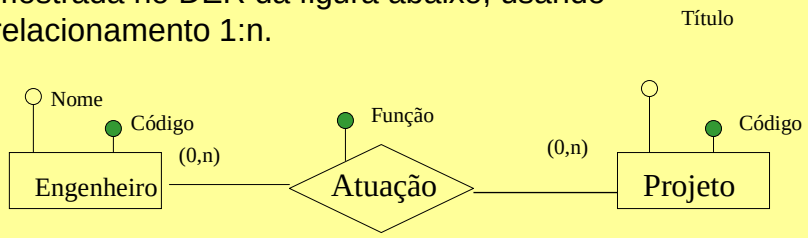
- O retângulo desenhado ao redor do relacionamento CONSULTA indica que este relacionamento passa a ser visto como uma entidade associativa.
- Sendo CONSULTA uma entidade, é possível associá-la através de relacionamentos a outras entidades.
- Alternativa:



57

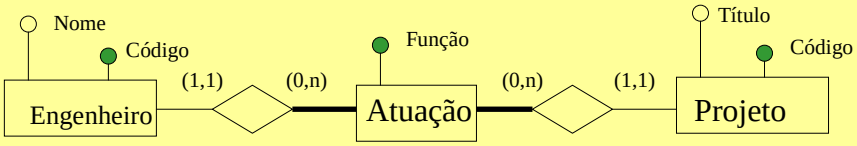
Exercícios Resolvidos

- Construa um DER que modela a mesma realidade mostrada no DER da figura abaixo, usando relacionamento 1:n.



58

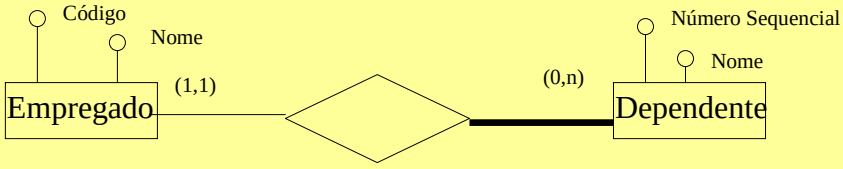
Solução



59

Exercícios Resolvidos

- Considere o relacionamento EMPREGADO-DEPENDENTE da figura abaixo. Considere que um dependente pode ser também empregado. Como o modelo deveria ser modificado para evitar o armazenamento redundante das informações das pessoas que são tanto dependentes quanto empregados?



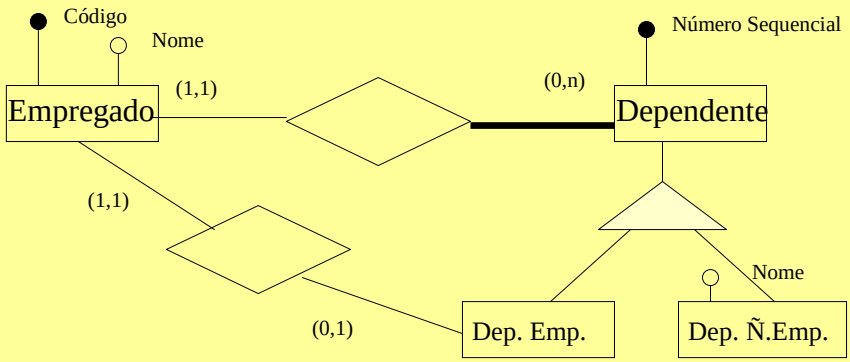
60

Solução

- A modificação consta em possibilitar que um dependente seja empregado.
- Caso se mantivesse o modelo original o nome do dependente seria armazenado redundantemente.
- A solução adotada foi a de especializar a entidade DEPENDENTE em duas que contem os atributos dos dependentes que não são empregados e DEP. EMP., que não contem atributos mas esta relacionada a entidade empregado correspondente.

61

DER - Solução



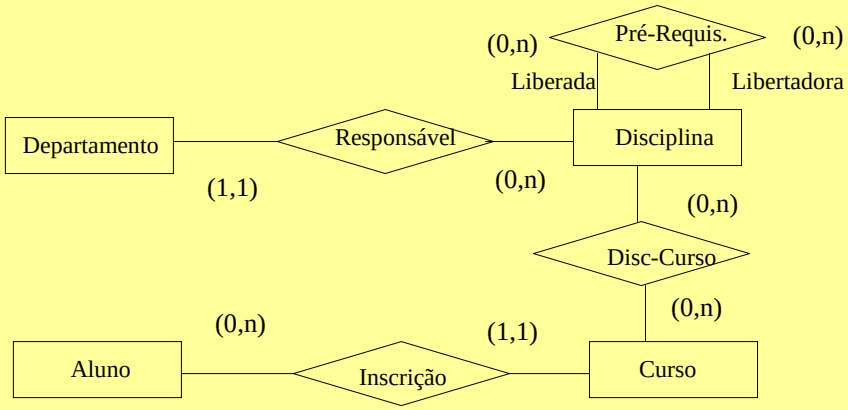
62

Exercícios

- Considere o DER abaixo. Modifique as cardinalidades mínimas de forma a especificar o seguinte:
  - Um curso não pode estar vazio, isto é, deve possuir ao menos uma disciplina em seu currículo;
  - Um aluno, mesmo que não inscrito em nenhum curso, deve permanecer por algum tempo no banco de dados.

63

Exercícios



64

Exercícios

1) Construa um diagrama E-R para uma companhia de seguros de automóveis com um conjunto de clientes, em que cada um possui um certo número de carros. Cada carro tem um número de acidentes registrados associados a ele.

2) Construa um diagrama E-R para um hospital com um conjunto de pacientes e um conjunto de médicos. Um registro de diversos teste realizados é associado a cada paciente.

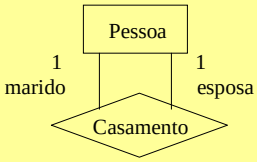
65

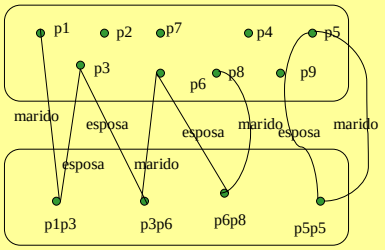
Propriedades do Modelo ER

- Um modelo ER é um modelo formal
  - Um modelo ER é formal, preciso e não ambíguo
  - Significa que diferentes leitores de um mesmo DER devem sempre entender exatamente o mesmo.
  - É fundamental um treinamento dos usuários e equipe de desenvolvimento para compreensão do DER.
- Abordagem ER têm poder de expressão limitado
  - São apresentadas apenas algumas propriedades de um banco de dados.
  - É uma linguagem muito pouco poderosa.
  - Para fazer com que o DER corresponda a realidade que deseja-se modelar, é necessário modificá-lo ou então definir restrições adicionais.

66

Exemplo



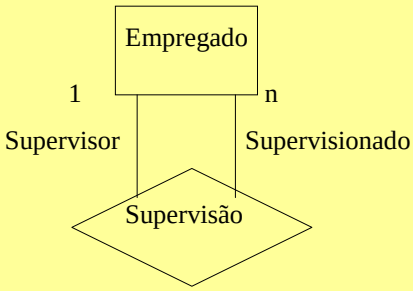


- O DER modela pessoas e casamentos e as ocorrências de PESSOA e CASAMENTOS.
- As ocorrências de CASAMENTOS não correspondem ao conhecimento da realidade.
- A pessoa P3 aparece em dois casamentos.
- A pessoa P5 aparece casada consigo mesma.
- A pessoa P6 aparece como marido em um casamento e esposa em outro casamento.

Características do DER

- É necessário lembrar o objetivo que se tem ao construir um DER é o de projetar um Banco de Dados.
- Neste contexto, o DER nada mais é do que uma descrição abstrata das estruturas do banco de dados.
- O objetivo do diagrama não é o de especificar todas restrições de integridade.
- Construções artificiais, isto é, construções incluídas no modelo apenas para satisfazer determinadas restrições de integridade são indesejáveis, pois distorcem os objetivos que se tem ao construir o DER.
- Há restrições de integridade que não se deixam incluir em um modelo ER.

Exemplo



Neste modelo é possível que um supervisionado seja um supervisor de seu superior

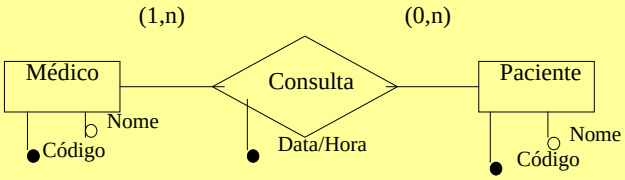
69

Modelos Diferentes podem ser Equivalentes

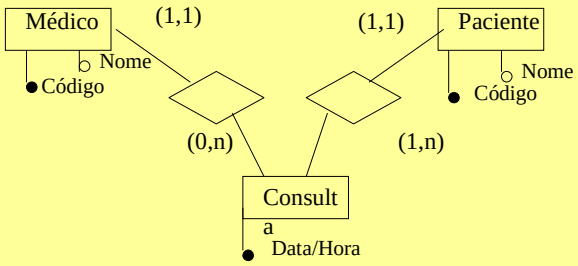
- Dois modelos são equivalentes, quando expressam o mesmo, ou seja quando modelam a mesma realidade.
- Para fins de projeto, dois modelos ER são equivalentes, quando ambos geram o mesmo esquema de Banco de Dados.

70

Consulta como Relacionamento n:n



Consulta como entidade



71

Passos para transformação de um relacionamento para entidade

1. O relacionamento n:n é representado como uma entidade.
2. A entidade criada é relacionada à entidades que originalmente participavam do relacionamento.
3. A entidade criada tem como identificador:
   - as entidades que originalmente participavam do relacionamento;
   - os atributos que eram identificadores do relacionamento original (caso o relacionamento original tivesse atributos identificadores)

72

Passos para transformação de um relacionamento para entidade

4 As cardinalidades da entidade criada nos relacionamentos de que participa é sempre (1,1).

5 As cardinalidades das entidades que eram originalmente associadas pelo relacionamento transformado em entidade são transcritas ao novo modelo.

73

Observações

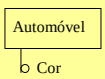
- Como um relacionamento n:n pode ser transformado em entidade é possível construir modelos sem relacionamentos n:n.
- Um outro caso de equivalência, é entre um modelo ER com um relacionamento de cardinalidade 1:1 e com a cardinalidade mínima 1 em ambos os lados, em um outro modelo ER em que tal relacionamento foi substituído por uma única entidade.

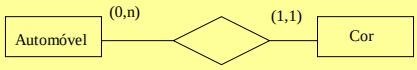
74

- A determinação de que construção da abordagem ER será usada para modelar um objeto de uma realidade considerada não pode ser feita através da observação do objeto isoladamente.
- É necessário conhecer o contexto, isto é, o modelo dentro do qual o objeto aparece.
- Não é aconselhável despender um tempo excessivo em longas discussões sobre como modelar um objeto.
- O próprio desenvolvimento do modelo e o aprendizado sobre a realidade irão refinando e aperfeiçoando o modelo.

75

Atributo x Entidade





- Como escolher se um objeto deve ser atributo ou entidade?
- Caso o objeto cuja modelagem está em discussão esteja vinculado a outros objetos (atributos, relacionamentos, entidades genéricas ou especializadas), o objeto deve ser modelado como entidade, já que um atributo não pode ter atributos, nem estar relacionado a outras entidades.

76

Atributo x Entidade

- Caso contrário deve ser modelado como atributo.
- Quando um conjunto de valores de um determinado objeto é fixo durante toda a vida do sistema ele pode ser modelado como atributo, visto que o domínio de valores de um atributo é imutável.
- Quando existem transações no sistema que alteram o conjunto de valores do objeto, o mesmo não deve ser modelado como atributo.

77

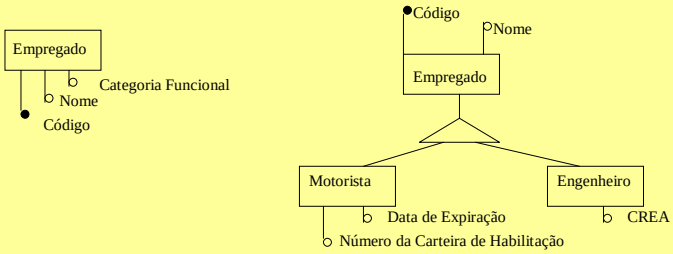
Atributo x generalização / especialização

- Uma especialização deve ser usada quando se sabe que as classes especializadas de entidades possuem propriedades (atributos, relacionamentos, generalizações, especializações) particulares.
- Faz sentido especializar e entidade empregado de acordo com a categoria funcional, no caso de classes particulares possuírem atributos ou relacionamento próprios.

78

Atributo x generalização / especialização

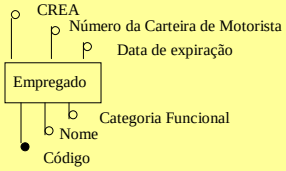
- Ainda no exemplo acima, o sexo do empregado é melhor modelar como atributo de empregado, caso não existam propriedades particulares de homens e mulheres a modelar na entidade considerada.



79

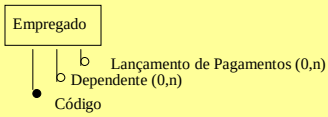
Atributos opcionais e multivalorados

- Quando se inicia o processo de modelagem é aconselhável tentar restringir-se ao uso de atributos obrigatórios e monovalorados.
- Em muitos casos, atributos opcionais indicam subconjuntos de entidades que são modelados mais corretamente através de especializações.
- Toda vez que aparecer um atributo opcional é aconselhável verificar se a modelagem através de entidades especializadas não é mais conveniente.



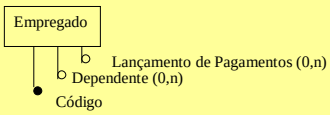
80

Atributos Multivalodados



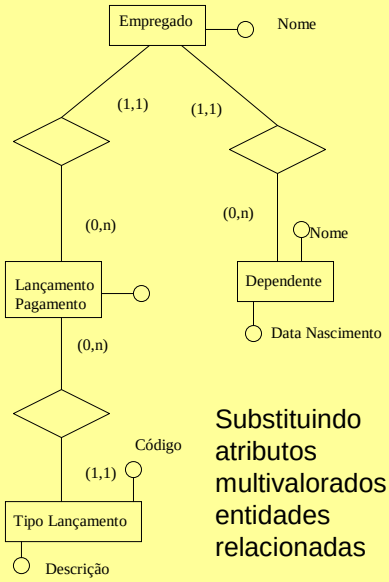
- Atributos multivalorados são considerados indesejáveis.
- Nos SGBD relacionais que seguem o padrão SQL/2, atributos multivalorados não possuem implementação direta.
- Alguns SGBD objeto/relacional já possuem implementação dos atributos multivalorados, padrão SQL/3.
- Atributos multivalorados podem induzir a um erro de modelagem, que é o de ocultar entidades e relacionamentos em atributos multivalorados.

81



Modelo utilizando atributos multivalorados

Ao considerar a entidade EMPREGADO mais detalhadamente, observa-se que tanto dependentes, quanto os lançamentos possuem propriedades particulares.



Substituindo atributos multivalorados por entidades relacionadas

82

O modelo deve ser correto

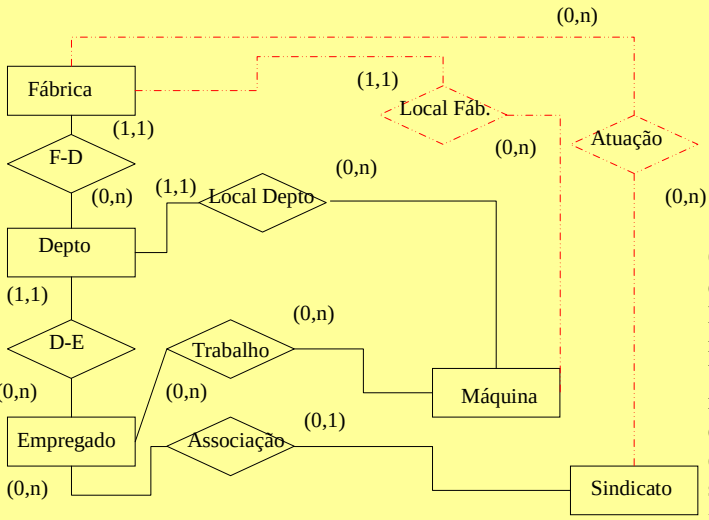
- Um modelo está correto quando não contém erros de modelagem, isto é, quando os conceitos de modelagem ER são corretamente empregados para modelar a realidade em questão.
- Pode-se distinguir entre dois tipos de erros:
  - Sintáticos
    - Quando o modelo não respeita as regras de construção de um modelo ER.
    - Associar atributos a atributos
    - Associar relacionamentos a atributos
    - Associar relacionamentos através de outros relacionamentos
  - Semânticos
    - Quando reflete a realidade de forma inconsistente.
    - Estabelecer associações incorretas
    - Usar uma entidade do modelo como atributo de outra entidade.
    - Usar o número incorreto de entidades em um relacionamento.

83

Modelo deve ser Completo

- Deve fixar todas propriedades desejáveis do banco de dados.
- Deve ser verificado por alguém que conhece profundamente o sistema a ser implementado.
- Deve-se verificar se todos os dados que devem ser obtidos do banco de dados estão presentes e se todas as transações de modificação do banco de dados podem se executadas sobre o modelo.
- Relacionamentos redundantes são relacionamentos que são resultado da combinação de outros relacionamentos entre as mesmas entidades.

84

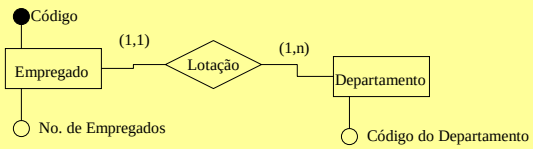


Os relacionamentos desenhados com linha pontilhada são redundantes.
Uma relacionamento é redundante, quando é possível elimina-lo do diagrama sem que haja perda de informações no banco de dados.

85

Outro tipo de redundância

- Atributos redundantes são atributos deriváveis a partir da execução de procedimentos de busca de dados e/ou cálculos sobre o banco de dados.



86

SQL

87

SQL

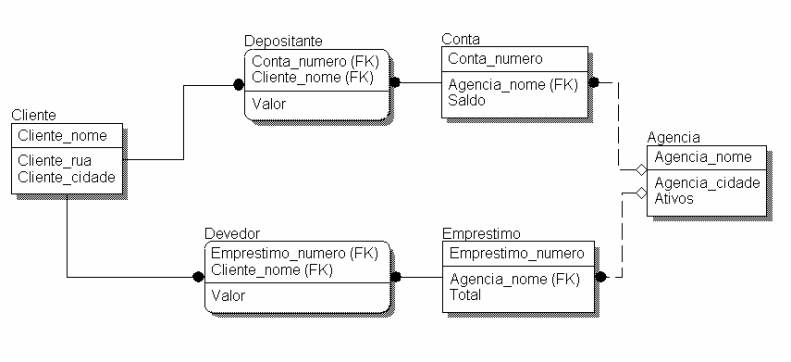
- Estrutura Básica
- Conjunto de Operações
- Funções Agregadas
- Valores Nulos
- Sub-consultas aninhadas
- Relações Derivadas
- Visões
- Modificação do Banco de Dados
- Relação de Junção (*Joined Relations)*
- Linguagem de Definição de Dados *(Data Definition Language)*
- SQL Embutido

88

Características do SQL

- Linguagem de quarta geração para auxiliar os programadores de aplicação para criar modelos de interface com usuário e formatar dados para relatórios.
- Disponível na maioria do Banco de Dados comerciais.
- A seção SQL -provê uma abstração do cliente para o servidor (que pode estar remoto).
  - Cliente pode se conectar ao servidor SQL, estabelecendo uma seção.
  - Executar uma série de comandos.
  - Desconectar a seção.
  - Pode confirmar ou desfazer o trabalho executado em uma seção.
- Um ambiente SQL contem inúmeros componentes, incluindo o identificador do usuário um esquema (schema).

89



90

Estrutura Básica

- SQL é baseado em conjuntos e operações relacionais com algumas modificações e melhoramentos.
- Uma consulta típica SQL tem a forma:

  select A1, A2, ... , An
  from r1, r2,...,rm
  where P

  – A - representa atributos
  – r - representa relações
  – P - é um predicado
- Esta consulta é equivalente em uma expressão da álgebra relacional:

$$\prod_{A_1,A_2,...,A_n,}(\sigma_P(r_1,r_2,...,r_m,))$$

- O resultado de uma consulta SQL é uma relação (tabela).

91

Cláusula SELECT

- A cláusula SELECT corresponde a operação de projeção da álgebra relacional. É utilizado para listar os atributos desejados como resultado da consulta.
- Encontrar os nomes de agências de todos os empréstimos:

  select agencia_nome
  from pagamento
- Na sintaxe pura da álgebra relacional, esta consulta é representada da seguinte forma:

$$\prod_{agencia\_nome}(pagamento)$$

- Um asterisco na cláusula select representa todos os atributos.

  select *
  from pagamento

92

Cláusula SELECT (continuação)

- SQL permite registros duplicados em relações como o resultado das consultas.
- Para forçar a eliminação dos registros duplicados, insere-se a palavra chave **distinct** depois de **select.**
- Encontrar os nomes de todas as agências da relação empréstimo e remover os registros duplicados.

```
select distinct agencia_nome
from pagamento
```

- A palavra chave **all** especifica que os registros duplicados não serão removidos

```
select all agencia_nome
from pagamento
```

93

Cláusula SELECT (continuação)

- A cláusula **select** pode conter expressões aritméticas envolvendo os operadores +, -, *, / e operados sobre constantes ou atributos das tuplas.
- A consulta:

```
select agencia_nome, emprestimo_numero, valor*100
from pagamento
```

- retorna uma relação equivalente à relação empréstimo com exceção que o atributo **valor** é multiplicado por 100.

94

Cláusula where

- A cláusula ***where*** corresponde em álgebra relacional ao predicado de seleção. Consiste em um predicado envolvendo atributos das relações que aparecem na clásula ***from.***
- Encontre todas os números de empréstimos efetuados na agência Portão com valor maior que 1200.

```
select emprestimo_numero
from pagamento
where agencia_nome = “Portão” and  valor > 1200
```

- SQL usa os conectores lógicos **and**, **or**, e **not**.
- Isto permite o uso de expressões aritméticas como operandos para os operadores de comparação.

95

Cláusula where (continuação)

- SQL inclui o operador de comparação **between** a fim de simplificar a cláusula **where** que especifica que um valor deve ser menor ou igual a algum valor e maior ou igual a outro valor.
- Encontrar o número do empréstimo das operações que envolvam valores entre 90000 e 100000.

```
select emprestimo_numero
from pagamento
where valor between 90000 and 100000
```

96

Cláusula FROM

- A cláusula **from** corresponde ao produto cartesiano da álgebra relacional. Ele lista as relações que devem ser avaliadas pelas expressões.
- Encontre o produto cartesiano entre o empréstimo e conta empréstimo.

  select from pagamento, emprestimo
- Encontrar o nome e o número de todos os clientes que possuem um empréstimo na agência do Portão.

  select distinct cliente_nome,pagamento.emprestimo_numero
  from pagamento, emprestimo
  where agencia_nome = “Portão” **and**
  pagamento.emprestimo_numero=emprestimo .emprestimo.numero

97

Operação Rename

- O mecanismo SQL para renomear relação e atributos é efetuado através da cláusula **as**:

  nome_velho **as** nome_novo
- Encontrar o nome e número do empréstimo de todos os clientes que possuam empréstimo na agência do Portão; alterar o nome da coluna *emprestimo_número* por *id*.

  select distinct cliente_nome, emprestimo.emprestimo_numero **as** id
  from pagamento, emprestimo
  where gencia_nome = “Portão” **and**
  pagamento.emprestimo_numero=emprestimo .emprestimo.numero

98

Tuplas Variáveis

- Tuplas variáveis são definidos na cláusula **from** através do uso da cláusula **as**.
- Encontre todos o nomes dos clientes e os números dos empréstimos para todos os clientes que possuam um empréstimo em alguma agência.

  select distinct cliente_nome,T.emprestimo_numero
  from pagamento **as** T, emprestimo **as** S
  where T.emprestimo_numero =S. emprestimo_numero
- Encontre os nomes de todas as agências que tem maior ativo que qualquer outra agência localizada em Curitiba.

  select distinct T. agencia_nome
  from agencia **as** T,agencia **as** S
  where T.ativos > S.ativos **and** S.agencia_cidade = “Curitiba”

99

Operadores String

- SQL inclui um operador de textos para comparação entre eles. Padrões são descritos usando dois caracteres especiais:
- porcentagem (%). O caracter % substitui qualquer texto.
- Sub barra (_). O caracter _ substitui qualquer caracter.
- Encontre os nomes de todos os clientes que possuam em seu endereço o texto Marechal.

  select cliente_nome
  from cliente
  where cliente_rua **like** “%Marechal%”
- Caso procure o palavra Marechal%

  like “Marechal \%”
  onde \ é o caracter de controle.

100

Ordenando a apresentação dos Registros

- Listando em ordem alfabética os nomes de todos os clientes que tenham um empréstimo na agência Perryridge.

```
select distinct cliente_nome
from pagamento, emprestimo
where pagamento.emprestimo_numero =
    emprestimo.emprestimo_numero and agencia_nome = “Perryridge”
order by cliente_nome
```

- Pode-se especificar **desc** para ordem descendente ou **asc** para ordem ascendente, para cada atributo. A ordem ascendente é o padrão quando não especificado.
- SQL deve executar uma ordenação completa para executar uma requisição **order by.** Uma vez que a ordenação de um grande número de registros pode ter um alto custo computacional, assim sugere-se ordenar somente quando necessário.

101

Operação de Conjunto

- As operações de conjunto **union, intersect** e **except** opera sobre relações e correspondem as operações da álgebra relacional $\cup$, $\cap$ e —.
- Cada uma das operações acima automaticamente elimina os registros duplicados. Para manter todos os registros duplicados deve-se utilizar os comandos **union all, insersect all** e **except all**. Suponha uma tupla ocorre *m* vezes em *r* e *n* vezes em *s*, então:

(m+n)	vezes em	*r* **union all** *s*
min(m,n)	vezes em	*r* **intersect all** *s*
max(0,m-n)	vezes em	*r* **except all** *s*

102

Exemplo

- Encontre todos os clientes que possuem conta e/ou empréstimo.

  (**select** cliente_nome **from** conta)
  union
  (**select** cliente_nome **from** emprestimo)
- Encontre todos os clientes que possuem um empréstimo e uma conta.

  (**select** cliente_nome **from** conta)
  intersect
  (**select** cliente_nome **from** emprestimo)
- Encontre todos os clientes que possuam conta mas não possuam empréstimo.

  (**select** cliente_nome **from** conta)
  except
  (**select** cliente_nome **from** emprestimo)

103

Funções Agregadas

- Estas funções operam sobre um conjunto de dados de uma coluna de uma relação e retornam um valor:

avg	valor médio
min	menor valor
max	maior valor
sum	soma dos valores
count	quantidade de valores

104

Exemplos

- Encontre a média dos ativos da agência Perryridge:
  select avg (ativos)
  from deposito
  where agencia_nome = “Perryridge”
- Encontre a quantidade de registros na relação clientes.
  select count (*)
  from cliente
- Encontre a quantidade de depositantes do banco.
  select count (**distinct** conta_numero)
  from deposito

105

Funções agregadas - group by

- Encontre o número de depositantes em cada agência
  select agencia_nome, **count** (**distinct** cliente_nome)
  from deposito, conta
  where deposito.conta_numero = conta.conta_numero
  group by agencia_nome
- Nota: Atributos na cláusula **select** fora da função de agregação deve aparecer na lista de atributos do **group by.**

106

Funções Agregadas - Having

- Encontro os nomes de todas as agências onde a média dos saldos é maior que $1200.

```
select agencia_nome,  avg (saldo)
from deposito
group by agencia_nome
having avg (saldo) > 1200
```

- Nota: predicados na cláusula **having** são aplicados depois dos grupos formados.

107

Valores Nulos

- É possível que algumas tuplas possuírem valores nulos, representados por **NULL**, para alguns atributos.
- Nulo (**NULL**) significa um valor desconhecido ou um valor que não existe.
- O resultado de qualquer expressão envolvendo nulo é **NULL**. De forma geral, todas as comparações envolvendo nulos retornam falso.

108

Comparação com nulos

- Qualquer comparação com nulo retorna **NULL.**
- (verdadeiro **or** nulo) = verdadeiro
- (falso **or** nulo) = nulo
- (nulo **or** nulo) = nulo
- (verdadeiro **and** nulo) = nulo
- (falso **and** nulo) = nulo
- (nulo **and** nulo) = nulo
- Resultado da cláusula **where** é tratado como falso se o que for avaliado for nulo.
- **P is null** retorna verdadeiro se **P** for nulo.

109

Exemplos

- Encontre todos os números de empréstimos que apresentam valores nulos no atributo valor.

```
select emprestimo_numero
from pagamento
where valor is null
```

- Total de todos os valores

```
select sum (valor)
from pagamento
```

- O comando acima ignora valores nulos, resulta em nulo se existirem somente valores nulos.
- Todas as operações, com exceção da operação **count(*)**, ignoram tuplas com valores nulos sobre os atributos agregados.

110

Consultas Aninhadas

- SQL prevê um mecanismo para aninhar sub-consultas.
- Uma subconsulta é uma expressão **select-from-where** que é aninhada com outra consulta.
- A sua utilidade é para executar testes para membros, comparações e cardinalidades de conjuntos,

111

Membros de Conjuntos

- F **in** r <=> $\Leftarrow$ t ε r (t =F)
- (5 in {0, 4, 5}) = Verdadeiro
- (5 in {0, 4, 6}) = Falso
- (5 not in {0, 4, 6}) = Verdadeiro

112

Exemplo

- Encontre todos os clientes que possuem uma conta e um empréstimo junto ao banco.

```
select distinct cliente_nome
from emprestimo
where cliente_nome in (select cliente_nome
from conta)
```

- Encontre todos os clientes que possuem um empréstimo mas não tem uma conta junto ao banco

```
select distinct cliente_nome
from emprestimo
where cliente_nome not in (select cliente_nome
from conta)
```


Exemplo

- Encontre todos os clientes que possuem um conta e um empréstimo junto a agência Perryridge.

```
select distinct cliente_nome
from pagamento, emprestimo
where pagamento.emprestimo_numero
   =emprestimo.emprestimo_numero  and agencia_nome = “Perryridge”
   and (agencia_nome, cliente_nome) in
(select agencia_nome, cliente_nome
   from deposito, conta
   where deposito.conta_numero = conta.conta_numero)
```

Conjunto de comparações

- Encontre todas as agências que possuem ativos maiores que algumas agências localizadas em Broklyn.

```
select distinct T.agencia_nome
from agencia as T,agencia as S
where T.ativos > S.ativos and
S.agencia_cidade = “Brooklyn”
```

115

Cláusula SOME

$$F <comp> some\ r \Leftrightarrow \exists t(t \in r \wedge [F <comp> t])$$

- Onde <comp> pode ser: <, ≤, >, ≥,=, ≠
- (5 < **some** { 0, 5, 6}) = verdadeiro
  - 5 é menor que alguma tupla da relação.
- (5 < **some** {0, 4, 5}) = falso
- (5 = **some** {0, 4, 5})= verdadeiro
- (5 ≠ **some** {0, 5}) = verdadeiro (pois 0 ≠ 5)
- (= **some**) ≡ **in**
- Porém, (≠ **some**) não é equivalente **not in**

116

Exemplo

- Encontre todas as agências que possuem ativos maior que algumas agências localizadas no Brooklyn.

```
select agencia_nome
from agencia
where ativos > some
   (select ativos
   from agencia
    where agencia_cidade = “Brooklyn”)
```

117

Cláusula ALL

$$F < comp > all\ r \Leftrightarrow \exists t(t \in r \wedge [F < comp > t])$$

- Onde <comp> pode ser: <, ≤, >, ≥ ,=, ≠
- (5 < **all** { 0, 5, 6}) = falso
  - 5 é menor que alguma tupla da relação.
- (5 < **all** {6, 10}) = verdadeiro
- (5 = **all** {0, 4, 5})= falso
- (5 ≠ **all** {0, 6}) = verdadeiro (pois 0 ≠ 5 e 6 ≠ 5)
- (≠ **all**) ≡ **not in**
- Porém, (= **all**) não é equivalente **in**

118

Exemplo

- Encontre todas as agências que possuem ativos maior que todas agências localizadas no Brooklyn.

```
select agencia_nome
from agencia
where ativos > all
   (select ativos
   from agencia
    where agencia_cidade = “Brooklyn”)
```

119

Testando relações vazias

- O construtor **exists** retorno o valor **verdadeiro** se o argumento da sub-consulta não é vazia.

$$\text{exists } r \Leftrightarrow r \neq 0$$

$$\text{not exists } r \Leftrightarrow r = 0$$

120

Exemplo

- Encontre todos os clientes que possuem contas em todas as agências localizadas para o Brooklyn.

```
select distinct S.cliente_nome
from deposito as S
where not exists (
   (select agencia_nome
    from agencia
    where agencia_cidade = “Brooklyn”)
    except
     (select R.agencia_nome
       from deposito as T, conta as R
       where T.conta_numero =R.conta_numero and
       S.cliente_nome =T.cliente_nome))
```

121

Teste da ausência de tuplas duplicadas

- O construtor **unique** teste se uma sub-consulta tem alguma tupla duplicada em seu resultado.
- Encontre todos os cliente que possuem somente uma conta na agência Perryridge.

```
select T.cliente_nome
from deposito as T
where unique (
   select R. cliente_nome
     from conta,deposito as R
     where T. cliente_nome =R. cliente_nome and
                R.conta_numero =conta. conta_numero and
                conta.agencia_nome = “Perryridge”)
```

122

Exemplo

- Encontre todos os clientes que possuem pelo menos duas contas na agência Perryridge.

```
select distinct T.cliente_nome
from deposito T
where not unique (
   select R. cliente_nome
     from conta, deposito as R
     where T. cliente_nome =R. cliente_nome and
              R.conta_numero =conta.conta_numero and
              conta.agencia_nome = “Perryridge”)
```

123

Relações Derivadas

- Encontre a média dos valores da conta daquelas agências onde a média dos valores é maior que $1200.

```
select agencia_nome, media_valor
   from (select agencia_nome, avg (valor)
              from conta
              group by agencia_nome)
              as result (agencia_nome, media_valor)
              where media_valor > 1200
```

- Não é necessário usar a cláusula **having**, desde que computa-se na cláusula **from** o resultados da relação temporária e os atributos do resultado podem ser utilizados diretamente na cláusula **where.**

124

Vistas (Views)

- Provê um mecanismo para esconder alguns dados de usuários.
- Para criar uma vista utiliza-se o comando:

create view v **as** <expressão de consulta>

- onde:

– < expressão de consulta > é qualquer expressão correta

– o nome da vista é designado em v.

125

Exemplo

- Uma vista com todas as agências e seus clientes.

```
create view todos_clientes as
    (select agencia_nome, cliente_nome, null
      from deposito,  conta
      where deposito.conta_numero =conta.conta_numero)
    union
    (select agencia_nome, cliente_nome
      from pagamento, emprestimo
      where  pagamento.emprestimo_numero=
             emprestimo.emprestimo_numero)
```

- Encontre todos os clientes da agencia Perryridge

```
select cliente_nome
    from todos_clientes
    where agencia_nome = “Perryridge”
```

126

Excluindo Registros

- Excluir todos os registros das contas da agência Perryridge.

```
delete from conta
where agencia_nome = "Perryridge"
```

- Excluir todas as contas de todas as agências localizadas na cidade de "CURITIBA"

```
delete from conta
   where agencia_nome in
     (select agencia_nome
     from agencia
     where agencia_cidade = "CURITIBA")
delete from deposito
   where conta_numero in (select conta_numero
     from agencia, conta
     where agencia_cidade = "CURITIBA"
     and branch.branch-name =account.branch-name)
```

127

Exemplo

- Excluir todos os registros de todos os depósitos com valor menor que a média do banco.

```
delete from deposito
   where valor < (select avg (valor)
   from deposito)
```

- Problema:
  - Quando exclui-se registros da tabela deposito, a média dos valores muda.
- Solução usado em SQL:

1 Calcular a média dos valores da tabela deposito.

2 Excluir todas as tuplas que satisfazem a condição sem recalcular a média.

128

Incluindo Registros

- Adicionando um novo registro em deposito.

```
insert into deposito
    values (“Perryridge”, “A-9732”, 1200)
```

- ou equivalente

```
insert into deposito
    (agencia_nome, valor, conta_numero)
    values
(“Perryridge”, 1200, “A-9732”)
```

- Adicionar uma nova tupla com valor nulo.

```
insert into deposito
    values (“Perryridge”, “A-777”,null)
```

129

Exemplo

- Providenciar um presente para todos os clientes de empréstimo da agência de Perryridge, depositando $200 em sua conta. Faça com que o número do empréstimo sirva como
- Faça com que o número do empréstimo sirva como o número da conta para novas contas.

```
insert into conta
    select cliente_nome, emprestimo_numero, null
      from emprestimo, pagamento
      where agencia_nome = “Perryridge” and
      emprestimo.emprestimo_numero=pagamento.emprestimo_numero
insert into deposito
    select agencia_nome,emprestimo_numero, 200
      from pagamento
      where agencia_nome = “Perryridge”
```

130

Atualização

- Aumente todas os depósitos com valores acima de $10.000 em 6% e as demais depósitos em 5%.
- Deve-se escrever dois comandos:

```
update deposito
set valor = valor* 1.06
where valor > 10000

update deposito
set valor = valor*  1.05
where valor <=  10000
```

- A ordem dos comandos é importante

131

Exemplo UPDATE

- Atualizando dados do cliente CAIO NAKASHIMA

```
update cliente set
   cliente_rua='Av. Sete de Setembro 3165',
   cliente_cidade = 'CURITIBA'
   where cliente_nome='CAIO NAKASHIMA'
```

Atualizando os ATIVOS de todas as agencia em 10%

```
update agencia set ativos=ativos*1,10;
```

132

Atualização de uma vista (view)

- Criar uma vista com todos os pagamentos da relação pagamento, escondendo o atributo valor.

```
create view agencia_emprestimo as
    select agencia_nome, emprestimo_numero
    from pagamento
```

- Adicionar um novo registro em agencia_emprestimo

```
insert into agencia_emprestimo
    values (“Perryridge”, “L-307”)
```

- Esta inserção deve representar uma inserção de uma tupla com os (“Perryridge”, “L-307”,null) na relação pagamento.
- Atualização em vistas complexas é complicado ou impossível de ser executado quando não é permitido.

133

Relação com junção (joined)

- As operações de junção pega duas relações e retorna como resultado uma outra relação.
- Estas operações são utilizadas como uma expressão de sub consulta na cláusula **from.**
- Condição de junção - define quais tuplas que nas duas relações se correspondem e quais atributos estarão presentes no resultado da junção.
- Tipos de junções - deinfe quantas tuplas de cada relação não se correspondem em outra relação devem ser consideradas.
  - **inner join**
  - **left outer join**
  - **right outer join**
  - **full outer join**

134

Exemplo

- Relação Pagamento

Agencia_nome	Emprestimo_numero	Valor
Downtown	L-170	$ 3.000,00
Redwood	L-230	$ 4.000,00
Perryridge	L-260	$ 1.700,00

- Relação Empréstimo

Cliente_nome	Emprestimo_numero
Jones	L-170
Smith	L-230
Hayes	L-155

135

Relação de Junção

pagamento **inner join** emprestimo **on** loan.loan-number =borrower.loan-number

Agencia_nome	Emprestimo_numero	Valor	Cliente_nome	Emprestimo_numero
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230

pagamento **left outer join** emprestimo **on** loan.loan-number =borrower.loan-number

Agencia_nome	Emprestimo_numero	Valor	Cliente_nome	Emprestimo_numero
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230
Perryridge	L-260	1700	null	null

136

Relação de Junção

pagamento **natural inner join** emprestimo

Agencia_nome	Emprestimo_numero	Valor	Cliente_nome
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith

pagamento **natural right outer join** emprestimo

Agencia_nome	Emprestimo_numero	Valor	Cliente_nome
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
null	L-155	null	Hayes

137

Relação de Junção

pagamento **full outer join** emprestimo **using** emprestimo_numero

Agencia_nome	Emprestimo_numero	Valor	Cliente_nome
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
Perryridge	L-260	1700	null
null	L-155	null	Hayes

- Encontre todos os clientes que possuem uma conta ou um empréstimo (mas não os dois) junto ao banco.

```
select cliente_nome
   from (deposito natural full outer join pagamento)
   where conta_numero is null or emprestimo_numero is null
```

138

SQL Embutido

- A norma define o SQL embutido em um grade número de linguagens como Pascal, PL/I, Fortran, C, COBOL, etc.
- Uma linguagem na qual SQL esta embutido é referido como a linguagem hospedeira e a estrutura do SQL embutido permite que a linguagem hospedeira compreenda os comandos SQL.
- A forma básica desta segue o padrão embutido no PL/I.
- O comando **EXEC SQL** é utilizado para identificar uma requisição de SQL embutido pelo preprocessador.

  EXEC SQL <comando SQL embutido > **END EXEC**

139

Exemplo

- A partir de uma linguagem hospedeira, encontre os nomes e número de contas de clientes com mais de um determinado valor em seu saldo. Este valor é o conteúdo de uma variável.

```
EXEC SQL
    declare c cursor for
    select cliente_nome, conta_numero
    from deposito, conta
    where deposito.conta_numero = conta.conta_numero
          and conta.saldo > :valor
END-EXEC
```

140

SQL Embutido (cont.)

- O comando **open** faz com que a consulta seja executada.

  EXEC SQL **open** c END-EXEC
- O comando **fetch** faz com o valor de uma tupla no resultado de uma consulta seja colocado em uma variável da linguagem hospedeira.

  EXEC SQL **fetch** c **into** :cn :an END-EXEC
- A chamada sucessiva do comando **fetch** retorna cada registro do resultado da consulta.
- Uma variável de comunicação SQL indica o final de arquivo.
- O comando **close** faz que seja excluída a tabela temporária do banco de dados que armazena o resultado da consulta.

  EXEC SQL **close** c END-EXEC

141

SQL Dinâmico

- Permite que programas construam e enviem consultas SQL em tempo de execução.
- Exemplo do uso de SQL dinâmico a partir de um programa em C.

```
char *sqlprog =“update conta set saldo =saldo * 1.05
      where conta_numero =?”;
EXEC SQL prepare dynprog from :sqlprog;
char conta[10] = “A-101”;
EXEC SQL execute dynprog using :conta
```

- O programa SQL dinâmico contem um **?**, que recebe valores quando o código SQL é executado.

142

Data Definition Language (DDL)

- Linguagem de Definição de Dados
- Permite a especificação de um conjunto de relações mas também informações sobre cada relação, incluindo:
- O schema (esquema) para cada relação.
- O domínio de cada valor associado com cada atributo.
- Restrição de Integridade.
- O conjunto de índices para ser mantido para cada relação.
- Informação sobre segurança e autorização para cada relação.
- A estrutura física de cada relação no disco.

143

Tipos de domínios em SQL

- **char(n)**. Texto de tamanho fixo, com tamanho especificado pelo usuário (n).
- **varchar(n)**. Texto de tamanho variável com tamanho máximo especificado pelo usuário (n).
- **int**. Inteiro (um subconjunto finito dos números inteiros que é dependente da plataforma).
- **smallint**. Inteiro pequeno (sub conjunto do domínio dos inteiros - *integer*, dependente da máquina).
- **numeric(p,d)**. Número real, ponto fixo, com a especificação do usuário de *p* dígitos com *n* dígitos à direita do ponto decimal.

144

Tipos de domínios em SQL (cont.)

- **real, double precision**. Ponto flutuante e precisão dupla com a precisão dependente de máquina.
- **float(n)**. Número de ponto flutuante, com a precisão especificada pelo usuário (n).
- **date**. Data contendo 4 dígitos para ano, mês e dia.
- **time**. Hora do dia em horas, minutos e segundos.
  - Valores nulos são permitidos em todos os tipos de domínios.
  - Declarando para ser **not null** proíbe-se valores nulos para o atributo.
  - Construtor **create domain** definindo em SQL-92, cria um tipo definido pelo usuário.

create domain person-name **char**(20) **not null**

145

CREATE TABLE

- Uma relação SQL é definido utilizado o comando **create table.**

  create table r (A_1 D_1,A_2 D_2,...,A_n D_n,
  h integrity-constraint $_1$,
  ...,
  h integrity- constraint $_k$)
  - r é o nome da relação
  - cada A_i é um nome de atributo da relação r.
  - D_i é um tipo de dado do domínio para o atributo A_i
- Exemplo:

  create table agencia
  (agencia_nome char(15) **not null**,
  agencia_cidade char(30),
  ativos integer)

146

Restrição de Integridade - Create Table

- **not null**
- **primary key** (A_1,...,A_n)
- **check** (P), onde P é um predicado
- Exemplo: Declarar *agencia_nome* como a chave primária para a agência e assegurar que os valores dos ativos não sejam negativos.

```
create table agencia
    (agencia_nome char(15) not null,
    agencia_cidade char(30),
    ativo integer,
    primary key (agencia_nome),
    check (ativo >=0))
```

- A declaração **primary key** sobre um atributo automaticamente assegura que não se nulo (**not null**) em SQL-92.

147

Construtores DROP e ALTER TABLE

- O comando **drop table** exclui toda informação sobre a tabela do banco de dados.
- O comando **alter table** é utilizado para adicionar atributos para uma tabela existente.
- Todas as tuplas da relação recebem o valor nulo para o novo atributo.
- O formato do comando **alter table** é:

  alter table r **add** A D
  - onde A é o nome do novo atributo da tabela;
  - D é o domínio de A.
- O comando **alter table** pode também ser utilizado para excluir atributos da relação.

  alter table r **drop** A
  - onde A é o nome da atributo da relação r.

148

Restrições de Integridade

- Restrição de Domínio
- Restrições de Integridade
- Assertions
- Gatilhos
- Dependências Funcionais

149

Restrição de Integridade

- Restrição de Integridade assegura contra danos acidentais sobre o banco de dados, assegurando que mudanças autorizadas sobre o banco de dados não resulte em perda da consistência dos dados.
- Restrição de domínio é a forma mais elementar de restrição.
- Eles testam se os valores inseridos no banco de dados e testam as consultas para garantir que as comparações fazem sentido

150

Restrição de Domínio (cont.)

- A cláusula **check** definida em SQL-92, permite que os domínios sejam restritos.
- Utiliza-se a cláusula **check** para assegurar que um valor maior que um valor específico.

  create domain hourly-wage **numeric(5,2)**

  constraint value-test **check**(value >=4.00)
- O domínio hourly-wage é declarado para ser um número decimal com 5 dígitos, 2 dos quais depois do ponto decimal.
- O domínio tem uma restrição que deve ser maior que 4,00.
- A cláusula **constraint** value-test é optativa, útil para indicar qual restrição foi violada.

151

Integridade Referencial em SQL

- Chave primária, candidata e estrangeira pode ser especificada como parte do comando SQL **create table.**
- A cláusula **primary key** do comando **create table** inclui uma lista de atributos que formam a chave primária.
- A cláusula **unique key** do comando **create table** inclui um lista de atributos que compõem a chave candidata.
- A cláusula **foreign key** inclui uma lista de atributos que formam a chave estrangeira e o nome da relação referenciada pela chave primária.

152

Exemplo

```
create table cliente
(   cliente_nome char(20) not null,
    cliente_endereco varchar(30),
    cliente_cidade varchar(30),
    primary key (cliente_nome))

create table agencia
(   agencia_nome char(15) not null,
    agencia_cidade varchar(30),
    ativos integer,
    primary key (agencia_nome))
```

153

Exemplo

```
create table conta
(   agencia_nome char(15),
    conta_numero char(10) not null,
    saldo integer,
    primary key (conta_numero),
    foreign key (agencia_nome) references agencia)

create table deposito
(   cliente_nome char(20) not null,
    conta_numero char(10) not null,
    valor number(10,2),
    primary key (cliente_nome, conta_numero),
    foreign key (conta_numero) references conta,
    foreign key (cliente_nome) references cliente)
```

154

Ações em Cascata

```
create table conta
    ...
    foreign key (agencia_nome) references agencia
    on delete cascade
    on update cascade,
    ...)
```

- A cláusula **on delete cascade** faz com que se excluído uma tupla da agência que resulta na violação da integridade referencial, exclua também as tuplas que se referem à agência que foi excluída.
- O comando de atualização funciona de forma similar.

155

Ações em Cascata (cont.)

- Se existe um elo de dependência de chave estrangeira através de relações múltiplas, com a especificação **on delete cascade** especificada para cada dependência, uma exclusão ou atualização que ocorre em um elo pode se propagar para toda corrente.
- Se uma atualização e exclusão em cascata causa a violação de restrição que não pode ser manipulada por uma operação em cascata o sistema aborta a transação.
- Como um resultado, todas as mudanças causadas por uma transação e ações em cascata são desfeitas.

156

Asserção

- Uma asserção é um predicado expressando uma condição que se deseja que o banco de dados sempre satisfaz.
- Uma asserção em SQL-92 tem a seguinte forma:

create assertion <nome_asserção> **check** <predicado>

- Quando um asserção é feita, o sistema testa sua validade.
- Este teste pode introduzir uma sobrecarga significativa no sistema, por isso asserções devem ser utilizadas com muito cuidado.

157

Exemplo

- A soma dos saldos dos empréstimos de cada agência deve ser menor que a soma de saldos de contas da agência.

```
create assertion verifica_emprestimo check
    (not exists
    (select * from agencia
      where (select sum (saldo) from emprestimo
      where emprestimo.agencia_nome =agencia.agencia_nome)
      >=(select sum(saldo) from conta
            where emprestimo.agencia_nome=agencia.agencia_nome)))
```

158

Exemplo

- Todo empréstimo pode ter pelo menos um cliente que manténha um conta com saldo mínimo de $1000,00.

```
create assertion verifica_saldo check
   (not exists
     (select * from emprestimo
      where not exists
              (select *
              from pagamento,deposito, conta
              where
emprestimo.emprestimo_numero=pagamento.emprestimo_numero
              and pagamento.cliente_nome=deposito.cliente_nome
              and deposito.conta_numero=conta.conta_numero
              and conta.saldo >= 1000)))
```

159

Gatilhos

- Gatilho é um comando que é executado automaticamente pelo sistema quando uma modificação ocorre no banco de dados.
- Para projetar um mecanismo de gatilho, deve-se:
- Especificar as condições sob as quais o gatilho deve ser executado.
- Especificar as ações que devem ser executadas pelo gatilho.
- A norma SQL-92 não inclui especificação de gatilhos, já esta implementado em muitos produtos.
- A norma SQL-3 já tem especificado a forma do gatilho.

160

Gatilho (cont.)

- Suponha que no lugar de permitir valores negativos para o saldo da conta, o banco trata o saque com:

– Ajusta o saldo a zero;
– Cria um empréstimo do valor do saque;
– Atribui o número do empréstimo igual ao número da conta que esta sofrendo um saque.

- A condição para executar o gatilho é uma atualização na relação conta que resulta em um saldo negativo.

161

Exemplo

```
define trigger limite on update of conta T
   (if new T.valor< 0
            then (insert into emprestimo.valor
            ( T.agencia_nome, T.conta_numero,- new T.saldo)
            insert into emprestimo
                     (select cliente_nome, conta_numero
                     from deposito
                     where T.conta_numero = deposito.conta-numero)
            update conta S
                     set S.saldo=0
                     where S.conta_numero = T.conta_numero))
```

- A palavra chave **new** utilizada antes do t.valor, indica que este é o valor inserido na tabela. Se **new** for omitido adota-se o valor antes da atualização.

162

SQL 3

- ISO/IEC 9075 : 1992
- SQL 3 - ISO/IEC 9075-x
- ISO/IEC 9075 -1 Framework
  - Descreve os conceitos fundamentais sobre os quais as demais partes de baseiam. Define também os termos, notações e convenções.
  - Especifica requerimentos de conformidade.
- ISO/IEC 9075 - 2 Foundation
  - Especifica os fundamentos de SQL
- ISO/IEC 9075 - 3 CLI - Call Level Interface
  - Especifica uma interface para SQL que pode ser usado por um programa de aplicação

163

SQL - 3 (cont.)

- ISO/IEC 9075 - 4 PSM - Persistent Stored Modules
  - Especifica estruturas de controle que possam ser usados por SQL Routines, e os módulos que pode contê-los.
- ISO/IEC 9075 - 5 Host Language Bindings
  - Especifica como comandos SQL podem ser embutidos em programas (host) e como podem ser preparados para execução.
- ISO/IEC 9075 - 6 XA Specialization (SQL Transaction)
  - Especifica como SQL pode ser usado com um gerenciador de transação.
- ISO/IEC 9075 - 7 Temporal
  - Especifica facilidades para definição e manipulação de dados Temporais.

164

SQL 3 (cont.)

- ISO/IEC 9075 - 9 MED- Management of External Data
  - Especificação de como SQL pode ser usado para gerenciamento de dados externos ao Banco de Dados
  - email, página HTML, documentos, planilhas, etc.
- ISO/IEC 9075 - 10 SQL / OLB
  - Object Language Binding
  - Especifica interface para linguagens OO tais como Java
- ISO/IEC 9075 - 11 SQL / OLAP
  - On Line Analytical Processing
  - Proposta: Especifica Funcionalidades para processamento analítico no SQL.
  - Enseja implicações paradigmáticas
    - Batch
    - OLTP
    - OLAP

165

O que muda?

- Novas funcionalidades para o SQL
- Mais regras de negócio para SGBD
- Aplicações mais leves
- Demanda Administrador de Dados mais forte
- Gerência de Metadados
  - Extended Types
  - Stored Procedures
  - Funções, Métodos,. Etc.
- Capacidade de Gerenciar dados fora do SGBD
  - Vídeo
  - Arquivos FTP
  - Arquivos de e-mail
  - etc.

166

Representação de várias formas de Dados Eletrônicos

- Texto
- Imagem
- Áudio
- Vídeo
- Impressão Digital
- GIS
- Séries Temporais

167

Objetos do SQL/3

- Objeto Linha (row object)
  - Basicamente uma tupla
- Abstract Data Type
  - Define que um objeto pode ser utilizado como componente de uma tupla.
- Declaração
  - CREATE ROW TYPE
  - nome do tipo
  - lista de atributos e seus tipos
- Sintaxe
  - CREATE ROW TYPE t (<declaração de componentes>)

168

Exemplo

- Fornecedor

```
CREATE ROW TYPE Tfornec
(  CodFornec        char(4),
   NomeFornec       varchar(40),
   EndFornec        Tend,
   StatusFornec     Integer
)
```

- Esta definição usa um tipo definido anteriormente (tend)
- Tuplas podem ser aninhadas

```
CREATE ROW TYPE Tend
(  RuaNo            varchar(60),
   Cidade           varchar(40),
   CEP              integer
)
```

169

Mesmo Exemplo (Oracle 8)

```
CREATE type Tend as object
( RuaNo   varchar(60),
  Cidade   varchar(40),
  CEP  integer
)

CREATE TYPE Tfornec as Object
(  CodFornec       char(4),
   NomeFornec      varchar(40),
   EndFornec       Tend,
   StatusFornec    Integer
)
```

170

Declarando Tabelas

- Sintaxe análoga a do SQL/2, usando
  OF ROW TYPE <nome do tipo linha>
- no lugar de lista de colunas da tabela.

- Exemplo

```
CREATE TABLE  FORNEC OF TYPE TFORNEC

CREATE TABLE  FORNEC OF TFORNEC (*)
```

- É possível definir o várias tabelas com o mesmo tipo de linha.

171

Acesso a campos de tuplas aninhadas

- Obter o código e a rua de cada fornecedor de Curitiba
- SELECT fornec.codfornec, fornec.EndFornec..RuaN
- FROM fornec
- WHERE fornec.EndFornec..Cidade = ‘CURITIBA’

172

HTML

173

Internet

- A Internet é uma rede mundial de redes de computadores.
- Podemos dizer que a Internet é a união dos computadores com as telecomunicações.
- Podemos dizer, também, que esta união tem causado grandes mudanças de comportamento econômico, social, e político em todo o mundo.
- A grande característica da Internet é proporcionar às pessoas a oportunidade de obter informações.
- Informações de qualquer natureza e localizadas em qualquer lugar.

174

Característica

- Sua arquitetura “aberta” é simples e baseada principalmente no compartilhamento de informações.
- Qualquer pessoa utilizando qualquer modelo de computador pode conectar-se à Internet.
- Atualmente atinge cerca de 50 milhões de usuários em todo o mundo.
- O crescimento da Internet no Brasil também é significativo.
- Dados recentes indicam que existem cerca de 50.000 computadores ligados à Internet no Brasil, e que este número tem crescido a uma taxa de 5.000 computadores por mês.

175

As aplicações básicas na Internet

- Copiar arquivos
- Enviar e receber mensagens eletrônicas
- Compartilhar informações
- Pesquisar por documentos e informações em geral
- Desenvolvimento de sistema para Tecnologia de Informações.

176

HTTP

- O Hypertext Transfer Protocol - ou HTTP como é chamado - é o protocolo através do qual as informações são transferidas utilizando o ambiente Web.
- A grande expansão da Web tem promovido este protocolo a um papel muito importante na Internet.
- Sua principal característica é a flexibilidade e a simplicidade.
- Assim, este protocolo é utilizado para transferir informações dos servidores para os clientes.
- Além do protocolo HTTP é preciso compreender o sistema de endereçamento dos servidores utilizando o Universal Resource Identifier (URI), que permite aplicações clientes localizarem servidores dentro da rede.

177

URI

- Universal Resource Identifier
- A medida que o número de protocolos cresce na rede, torna-se necessário uma padronização e simplificação da maneira como um determinado serviço é identificado e endereçado na rede.
- Os URIs permitem a existência de uma série de mecanismos de endereçamento.
- O URI define um método para “empacotar” um nome de um objeto de dado com um endereço universal pelo qual ele pode ser localizado no campo de outros endereços universais.
- O sistema de endereçamento não necessariamente revela qualquer informação útil a respeito do objeto de dado básico, sua metodologia de acesso, ou o sistema no qual o dado reside.
- O URI meramente permite que o dado seja encontrado.

178

URL

- Uniform Resource Locator (URL)
- O URI pode incluir informações que podem identificar qual o princípio de acesso.
- Este tipo de identificador é chamado Uniform Resource Locator ou URL como ficou conhecido na terminologia da Internet.
- Assim, o URL é um URI contendo informações adicionais sobre como o objeto que está sendo endereçado deve ser acessado.

179

HTML

- HTML é uma linguagem de formatação de textos utilizada para definir páginas na Web, baseada em códigos embutidos em um documento que pode servir para ajustar fontes, criar listas, mostrar imagens, entre outros tipos de formatações de páginas.
- A linguagem HTML está fundamentada na ISO **S**tandard **G**eneralized **M**arkup **L**anguage (ou SGML como é conhecida), que é um padrão internacional de formatação de documentos.
- Como um subconjunto deste padrão ISO, qualquer aplicação que possa interpretar o formato SGML poderá também ler o formato HTML.

180

HTML

- Como o próprio nome diz, a HTML é uma linguagem de marcação hipertexto, e é considerada a “linguagem da Web”.
- Todos os documentos que você acessar na Web foram implementados em HTML por alguém.
- As formatações de páginas, as imagens coloridas, os “hyperlinks” que possibilitam “navegar” pelo mundo virtual, foram desenvolvidos utilizando HTML.
- A linguagem HTML é fácil de aprender e requer na verdade muita criatividade.
- Basicamente, os documentos escritos em HTML são arquivos no formato ASCII-texto.
- Assim, podem ser criados com a utilização de qualquer editor de texto que grave os arquivos em formato texto “puro”.

181

Característica

- HTML é uma linguagem simples, porém poderosa e com muitos recursos.
- Um fator importante para o seu aprendizado é compreender seus conceitos, seus propósitos, e o que ela pode nos oferecer.
- Algumas das principais características da HTML são :
  - Formatação de documentos
  - Organização de listas
  - Capacidade de incluir hipertexto/hipermídia em documentos Web
  - Capacidade de incluir imagens clicáveis.

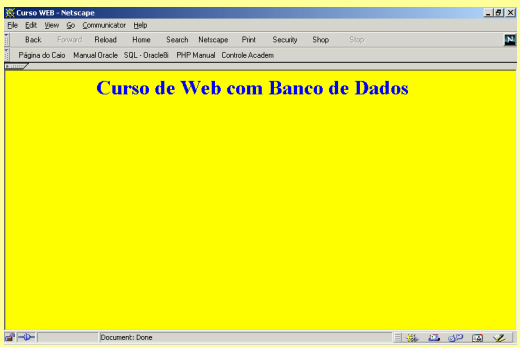
182

Compreendendo a estrutura HTML

- A HTML faz uso de “tags” e elementos que proporcionam uma maneira de especificar qual formato utilizar e quando um formato começa e termina.
- Para isso, utiliza *marcações* específicas e distintas para dizer ao browser do usuário como exibir um documento.
- A sintaxe básica para estas marcações HTML geralmente são especificadas da seguinte maneira:

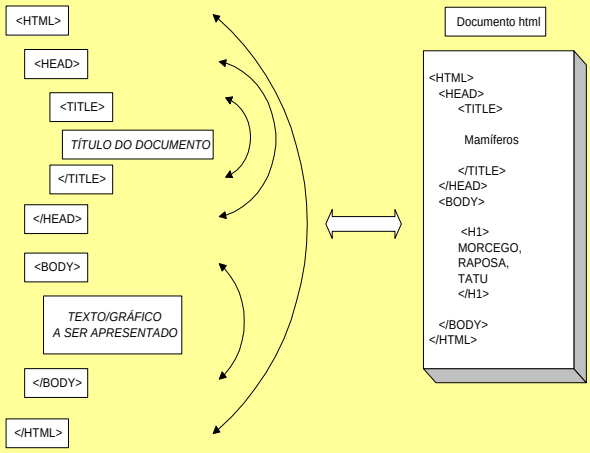
183

```
<HTML>
   <HEAD>
        <TITLE> Curso WEB </TITLE>
        <META name=“description” content=“Página do CAIO”>
        <META name=“keyword” content=“web, banco de dados”>
   <HEAD>
   <BODY bgcolor=yellow text=blue  link=read vlink=green>
         <CENTER>
        <H1> Curso de Web com Banco de Dados </H1>
   </BODY>
</HTML>
```



184

Estrutura de um documento HTML



Cabeçalho indica títulos, refresh, informações para pesquisa
Corpo indica todos os textos, imagens que serão mostradas

185

TAG

- O tag determina a formatação do elementos e a forma como os mesmos serão apresentados pelo browser.
- O tag é delimitado pelos sinais ”<“ e “>“ . O tag geralmente é utilizado aos pares a fim de delimitar o início e o fim de uma formatação. O **fim** de umtag é indicado pelo caracter “/ “ antes da definição da formatação.
- Tags vazias realizam sua função por conta própria, têm um sentido sozinhas.

 cria uma quebra de linha em qualquer ponto em que for usada.
  <HR> cria um linha na página

186

TAGs Básicos

<HTML> </HTML>

Indica que o arquivo contém dados no formato HTML. Todos os elementos e formatações devem estar dentro desse TAG.

<HEAD> </HEAD>

Indica a área de cabeçalho do documento. Dentro dele devem estar o título do documento e outras informações referentes ao documento.

<BODY atributos> </BODY>

Indica a área dentro da qual todos os elementos (imagem, texto e endereços estão. São os elementos desse local que o browser irá mostrar para o usuário.

background="url_imagem" - imagem que será usada como fundo.
text="#nnnnnn" - cor default do texto no formato RGB
link = "#nnnnnn" - cor para os links da página
vlink = "#nnnnnn" - cor para os links já visitados

187

Cabeçalho

- **<BASE>**

  Tem um atributo de nome HREF que pode incluir um endereço usado como diretório de partida para suas imagens e vínculos. Pode ser utilizado para referenciar, qual é seu provedor.

  Dentro do arquivo HTML, assim, quaisquer diretórios ou arquivos pedidos por vínculos ou imagens serão tidos como pertinentes ao endereço fornecido em:

  <base href="http://www.cefetpr.br">

188

Cabeçalho

- **<META>**

  Possui atributos “name” e “content”.

  <meta name=“description” content=“Página do CAIO”>

  descreve as intenções da sua página ou site.

  <meta name=“keyword” content=“web, banco de dados”>

  descreve as palavras chaves que serão utilizados por sites de pesquisa como Alta Vista ou Yahoo.
- **<TITLE> </TITLE>**

  Dentro deles, deve-se especificar o título do documento. Esse dado geralmente mostrado na parte superior do browser.

189

Estilos de Textos

- <B> **NEGRITO** </B>
- <I> *ITÁLICO* </I>
- <U> SUBLINHADO </U>
- ^{Sobre ^{escrito}}
- _{Sub $_{escrito}$}

190

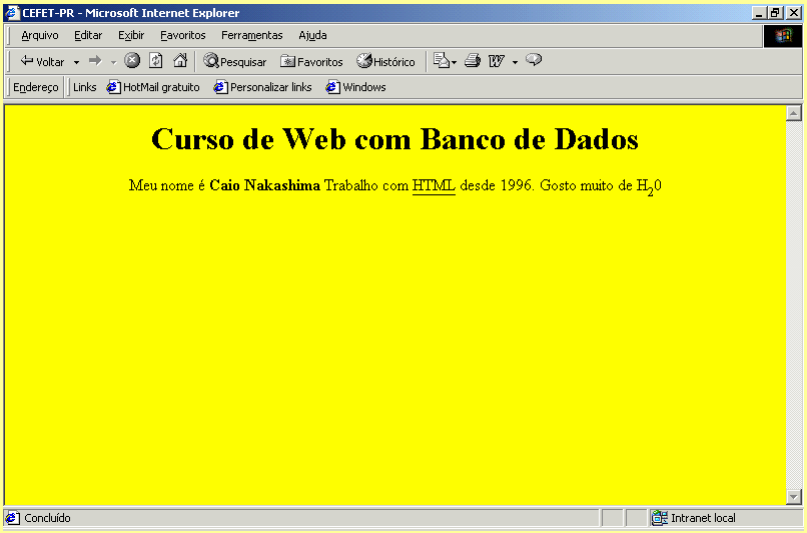
Exercício

- Desenvolva uma página HTML contendo
  - Título: Nome de sua empresa
  - Palavras Chaves: Atividades de sua empresa
  - Cor de fundo: Amarelo
  - Texto: Preto
  - Cor de Link: Branco
  - Cor de Link Visitado: Vermelho
  - Os dados básicos de sua atividade utilizando pelo menos 3 Tag de estilo.
  - Grave em disco e veja o resultado.

191

```
<HTML>
   <HEAD>
        <TITLE> CEFET-PR </TITLE>
        <META name="description" content="Página do CAIO">
        <META name="keyword" content="web, banco de dados">
   <HEAD>
   <BODY bgcolor=yellow text=black  link=white vlink=red>
         <CENTER>
        <H1> Curso de Web com Banco de Dados </H1>
        Meu nome é <B> Caio Nakashima </B>
        Trabalho com <U>HTML</U> desde 1996. Gosto muito de
   H<sub>2</sub>0
   </BODY>
</HTML>
```

192



193

Texto

- **<Hn> </Hn>**

  Indica títulos (em negrito e fonte de letra diferenciada) dentro do documento, sendo que “n” deve indicar o tamanho do título, que pode ser de 1 até 6 e onde o 1 indica a maior fonte.

  Uma linha extra separará o cabeçalho do resto do texto.

- **
**

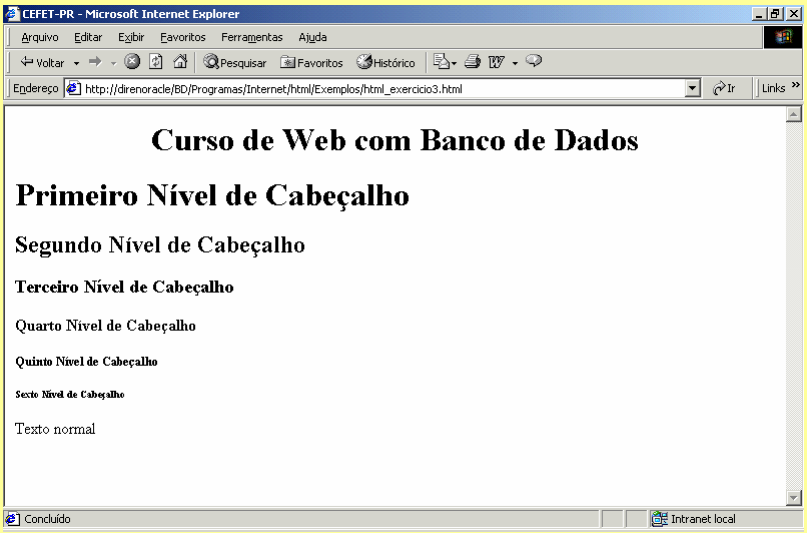
  Este TAG força a quebra de linha de um texto ou elemento. Devido a sua função não requer TAG de fechamento.

  “BReak”

194

```
<HTML>
    <HEAD>
    <TITLE> CEFET-PR </TITLE>
          <META name="description" content="Página do CAIO">
          <META name="keyword" content="web, banco de dados">
    <HEAD>
    <BODY bgcolor=white text=black  link=blue vlink=red>
     <CENTER>
          <H1> Curso de Web com Banco de Dados </H1>
          </CENTER>
          <H1>Primeiro Nível de Cabeçalho</H1>
          <H2>Segundo Nível de Cabeçalho</H2>
          <H3>Terceiro Nível de Cabeçalho</H3>
          <H4>Quarto Nível de Cabeçalho</H4>
          <H5>Quinto Nível de Cabeçalho</H5>
          <H6>Sexto Nível de Cabeçalho</H6>
          Texto normal
    </BODY>
</HTML>
```

195



196

Parágrafos

- **<P> </P>**

  A marcação de parágrafo é utilizada para definir o início de um novo parágrafo.

  Entre dois parágrafos é deixado uma linha em branco.

  Portanto, este tipo de marcador é utilizado para separar a informação entre blocos lógicos de texto.

  A linguagem HTML não reconhece o caracter de quebra de linha dos editores de texto.

  Mesmo que exista uma linha em branco, os clientes Web só reconhecem o início de um novo parágrafo mediante a marcação apropriada.

197

Textos

- **<PRE>...</PRE>**

  Texto Pré-Formatado. A marcação <PRE>...</PRE> é utilizada para representar blocos de texto com suas formatações originais, ou seja, espaços em branco, tabulações, e quebras de linhas são preservados.

- **<EM>...</EM>**

  Ênfase. É utilizada para a formatação de textos enfatizados, representados com a utilização de fontes itálicas ou negritadas.

- **<STRONG>...</STRONG>**

  desempenha função parecida utilizando fontes negritadas.

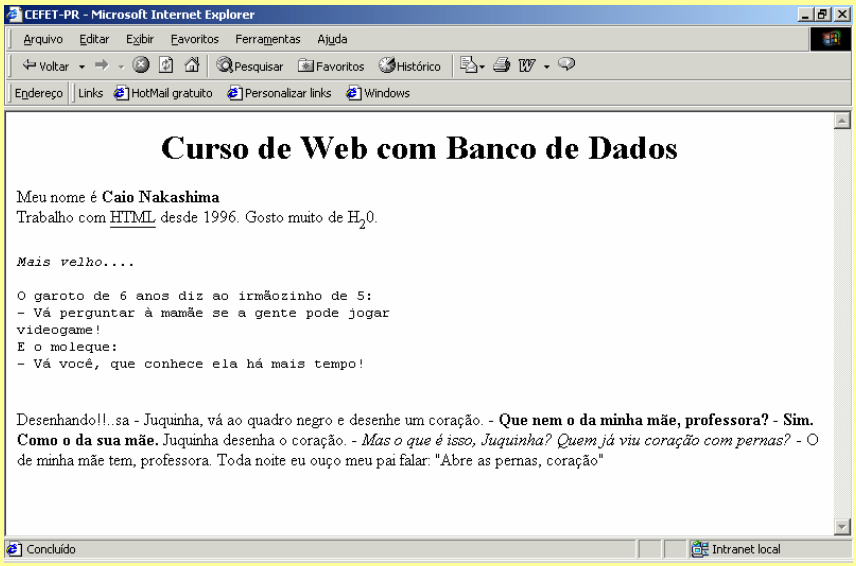
- **<CITE>...</CITE>**

  é utilizada para a formatação de citações, geralmente utilizando fontes itálicas.

198

```
<HTML>
    <BODY bgcolor=white text=black  link=blue vlink=red>
    <CENTER>
        <H1> Curso de Web com Banco de Dados </H1>
        </CENTER>
        Meu nome é <B> Caio Nakashima </B><BR>
        Trabalho com <U>HTML</U> desde 1996. Gosto muito de H<sub>2</sub>0.
<PRE>
<EM>Mais velho....</EM>
O garoto de 6 anos diz ao irmãozinho de 5:
- Vá perguntar à mamãe se a gente pode jogar
videogame!
E o moleque:
- Vá você, que conhece ela há mais tempo!
</PRE>
<blockquote>
Desenhando!!..
- Juquinha, vá ao quadro negro e desenhe um
coração.
<STRONG>- Que nem o da minha mãe, professora?
- Sim. Como o da sua mãe.</STRONG>
Juquinha desenha o coração.
<CITE>- Mas o que é isso, Juquinha? Quem já viu coração
com pernas?</CITE>
- O de minha mãe tem, professora. Toda noite eu ouço
meu pai falar: "Abre as pernas, coração” </blockquote>
    </BODY>
</HTML>
```

199



200

Textos

- **<BLOCKQUOTE> </BLOCKQUOTE>**

Faz com que todo o texto apareça uniformemente recuado.

- **<FONT> </FONT>**

FACE = tipo da fonte

SIZE = tamanho da fonte

COLOR = cor da fonte

- **<HR atributos>**

Faz com que o browser desenhe uma linha horizontal.

Atributos:

size=n indica a espessura da linha onde n é um número.

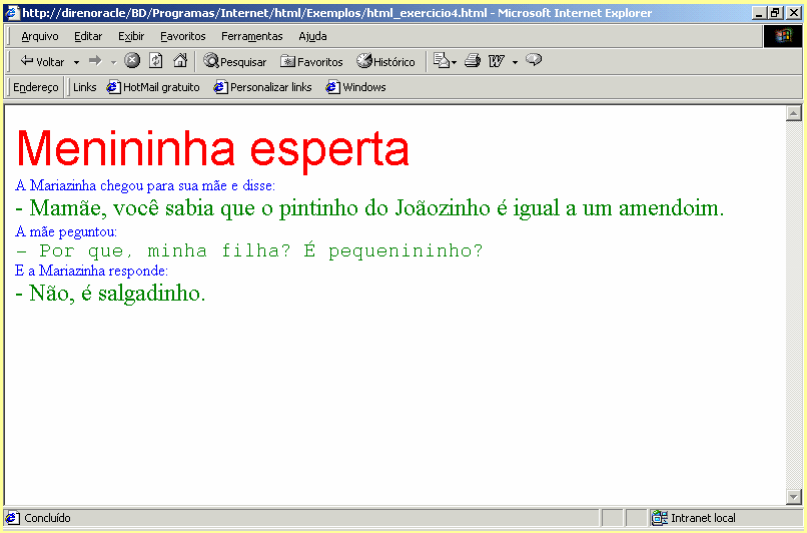
width="n%" indica a largura ocupada na tela .

201

```
<HTML>
    <BODY bgcolor=white text=black  link=blue vlink=red>
    <FONT color=Red size=7 face=ARIAL>Menininha esperta </FONT>

    <BR><FONT color=Blue size=3>
    A Mariazinha chegou para sua mãe e disse: </FONT>
    <BR><FONT color=Green size=+2>
    - Mamãe, você sabia que o pintinho do
    Joãozinho é igual a um amendoim.          </FONT>
    <BR><FONT color=Blue size=3>
    A mãe peguntou:
    <BR><FONT color=Green size=+2 face=COURIER>
    - Por que, minha filha? É pequenininho? </FONT>
    <BR><FONT color=Blue size=3>
    E a Mariazinha responde: </FONT>
    <BR><FONT color=Green size=+2>
    - Não, é salgadinho.</FONT>
    </BODY>
</HTML>
```

202



203

Listas

- **<UL> </UL>**

Isso indica a existência de uma lista não numerada.

Cada item da lista é indicado pelo TAG “<LI>“ (list item).

ex: ...

```
<UL>
    <LI> Pilha
    <LI> Bateria
    <LI> Acumulador
</UL>
```

204

Listas Ordenadas

- **<OL> </OL>**

Isso indica a existência de uma lista numerada.

Cada item da lista é indicado pelo TAG “<LI>“ (list item).

ex: ...

```
<OL>
   <LI> Caixa
   <LI> Prego
   <LI> Serra
</OL>
```

205

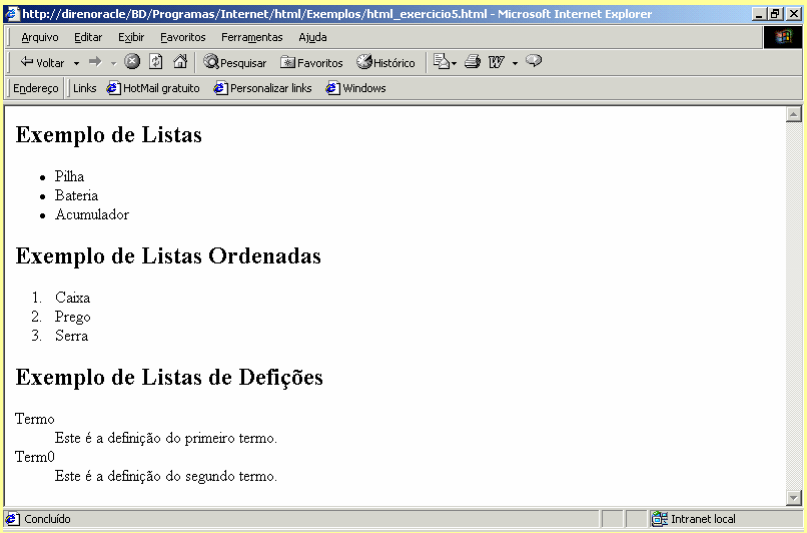
Listas de Definições

- Uma lista de definições permite incluir uma descrição de cada item listado.
- Por isso, alguns autores chamam este tipo de lista de ***lista de glossário***.
- Uma lista de definição utiliza o marcador **<DL>**.
- Normalmente consiste de um termo (através da marcação **<DT>)** e de uma definição (através da marcação **<DD>)**.
- Os browsers clientes geralmente formatam a definição em uma nova linha com outro alinhamento.

206

```
<HTML>
    <BODY bgcolor=white text=black  link=blue vlink=red>
    <H2> Exemplo de Listas </H2>
    <UL>
            <LI> Pilha
            <LI> Bateria
            <LI> Acumulador
  </UL>
    <H2> Exemplo de Listas Ordenadas</H2>
    <OL>
       <LI>  Caixa
       <LI> Prego
       <LI>  Serra
    </OL>
    <H2> Exemplo de Listas de Defições</H2>
    <DL>
            <DT>Termo<DD>Este é a definição do primeiro termo.
            <DT>Termo<DD>Este é a definição do segundo termo.
    </DL>
    </BODY>
</HTML>
```

207



208

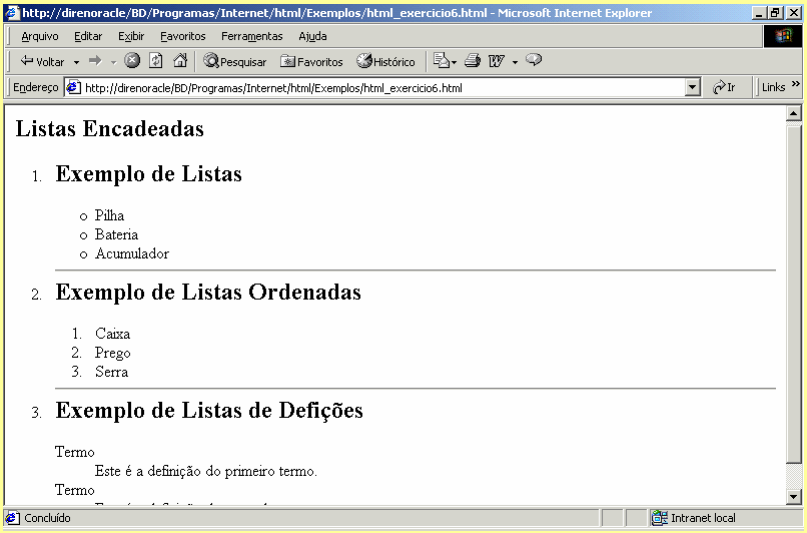
Lista Encadeadas

- As listas podem ser sucessivamente encadeadas (ou seja, uma lista dentro de outra lista), embora uma boa prática seja você limitar o encadeamento a três níveis no máximo.
- Com isto você conseguirá produzir resultados satisfatórios e facilitará a compreensão por parte do leitor da sua página Web.
- Por exemplo, você poderá ter um parágrafo intercalado com uma lista que contenha um único item.

209

```
<HTML>
    <BODY bgcolor=white text=black  link=blue vlink=red>
    <h2>Listas Encadeadas</h2>
    <OL>
          <LI><H2> Exemplo de Listas </H2>
                <UL>
                      <LI> Pilha
                      <LI> Bateria
                      <LI> Acumulador
          </UL><hr>
          <LI><H2> Exemplo de Listas Ordenadas</H2>
                <OL>
                      <LI>  Caixa
                      <LI> Prego
                      <LI>  Serra
                </OL><hr>
          <LI><H2> Exemplo de Listas de Defições</H2>
                <DL>
                      <DT>Termo<DD>Este é a definição do primeiro termo.
                      <DT>Termo<DD>Este é a definição do segundo termo.
                </DL>
     </OL>
    </BODY>
</HTML>
```

210



211

Caracteres Especiais no HTML:

Caracter Especial	Representação HTML
<	<
>	>
&	&
“	"
º	º
Espaço	

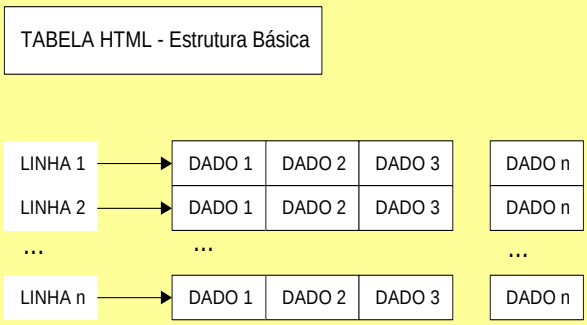
Caracter acentuado	Representação HTML	Significado
‘	&_acute;	acento agudo
`	&_grave;	acento grave
^	&_circ;	circunflexo
~	&_tilde;	til
,	&_cedil;	cedilha

Observações:
Nos acentos, a primeira letra após o “&“ é a letra que será acentuada.
Nesses caso, o HTML faz diferenciação entre maiúsculas e minúsculas.
Os browsers mais atuais permitem a acentuação direta da palavra.

212

TABELAS HTML

As tabelas no HTML são apenas uma maneira de se mostrar dados de forma tabular (não têm qualquer relação com armazenamento de dados).



213

<TABLE atributos> </TABLE>

- Indica o início e o fim de uma tabela. Dentro desse TAG devem existir outros TAG de tabela para a formatação da mesma.
- Atributos:
  - align= “CENTER || LEFT || RIGHT” - posição da tabela em relação à janela do navegador.
  - border=n - indica a espessura para a borda da tabela
  - bordercolor =“#nnnnnn” - cor da borda
  - bgcolor=“#nnnnnn” - indica a cor de fundo para toda a tabela.
  - cellspacing=n - espaço entre cada célula de uma tabela.
  - cols = n se
  - background=“arquivo” - imagem no fundo da tabela.
  - width=“n%” - indica a largura da tabela em relação ao tamanho da janela do browser, p.
  - valign=“TOP || BOTTOM || CENTER”

214

Código de Cores

COR		RED	GREEN	BLUE	
Branco	#	FF	FF	FF	#FFFFFF
Azul	#	00	00	FF	#0000FF
Preto	#	00	00	00	#000000
Verde	#	00	80	00	#008000
Verde-limão	#	00	FF	00	#00FF00
Vermelho	#	FF	00	00	#FF0000
Amarelo	#	FF	FF	00	#FFFF00

215

Cabeçalho de Tabela

<CAPTION> </CAPTION>

- Título da tabela. Dentro de uma tabela só deve existir um único título definido pelo texto entre esse TAG.

<TH atributos> </TH>

- Especifica o título de uma célula. Eqüivale ao <TD> </TD>, porém escreve o texto em negrito. atributos:
- possui os mesmos atributos de <TD>.

<TR atributos> </TR>

- Indica início e fim de uma linha da tabela atributos:
  - bgcolor="#nnnnnn" - indica a cor para a linha da tabela. (tem preferência sobre a cor de fundo da tabela)

216

TAG de tabelas

<TD atributos> </TD>

- Determina o início e o fim de um dado (ou célula) da tabela.Note que o conteúdo da célula pode ser outra linha de tabela, texto pré-formatado, ...
- Atributos:
  - nowrap - indica que o conteúdo da célula deve ser apresentado em uma única linha
  - rowspan = n - indica o número de linhas que esta célula irá ocupar
  - colspan = n - indica o número de colunas que esta célula irá ocupar
  - bgcolor="#nnnnnn" - indica a cor de fundo da célula (tem preferência sobre a cor de fundo da linha)

217

```
<HTML>
    <HEAD> <TITLE> CEFET-PR </TITLE> <HEAD>
    <BODY bgcolor=yellow text=black  link=red vlink=red>
     <CENTER>
      <H1> Curso de Web com Banco de Dados </H1>
          <table bgcolor="#00FFFF" border=10 width=100%>
                    <caption> Avaliação da Aula </caption>
                    <th width=40% rowspan=2> Item</th>
                    <th colspan=3> Avalia&ccedil;&atilde;o</th>
                    <tr>
                              <th width=20%>Regular</th>
                              <th width=20%>Bom</th>
                              <th width=20%>&Oacute;timo</th>
                    </tr><tr>
                              <td bgcolor="white" nowrap>Coffee-Break</td>
                              <td> </td>
                              <td> </td>
                              <td> </td>
                    </tr><tr>
                              <td bgcolor="white">Computadores</td>
                              <td> </td>
                              <td> </td>
                              <td> </td>
                    </tr>
          </table>
    </BODY>
</HTML>
```

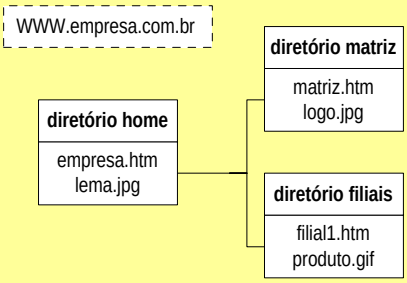
218

219

Caminhos de arquivos

- Quando se faz referência a outros arquivos em um documento HTML, pode-se especificar o caminho completo ou relativo do mesmo.
- Ao se especificar o caminho completo costuma-se indicá-lo na forma padrão de URL.
- Ex: “http://www.empresa.com.br/matriz/matriz.htm”.
- Ao se especificar um caminho relativo, deve-se considerar sempre que o diretório atual (o mostrado na URL) é o ponto de partida.
- ex supondo que a página atual é a empresa.htm e que se queira relacioná-la com a imagem logo.jpg do diretório matriz:
- “matri/logo.jpg”.

220



ex: supondo que a página atual seja a filial1.htm e se queira referenciar a imagem logo.jpg , o caminho seria: "../matriz/logo.jpg"

221

Vínculos

<A HREF="url_destino"> texto </A>

- Ao se clicar com o mouse nesse link, a nova URL é carregada pelo browser.
  - texto - indica o texto que servirá de referência para indicar o link (costuma conter um breve relato da página destino, ou uma palavra-chave da mesma).
  - url_destino - Pode indicar a URL do documento a ser chamado (pode ser um caminho absoluto ou relativo).
  - caso a url_destino seja:**"mailto:webmaster@empresa.com.br"** o browser irá abrir um e-mail para o o endereço referido.
  - caso seja: **"#nome"** irá para o local do mesmo documento que tenha o TAG **<A name="nome"> </A>,** onde os nomes sejam iguais.

222

Imagens e links

<IMG SRC="local_da_imagem" atributos>

- Mostra a imagem indicada pelo local_da_imagem". O arquivo de imagem deve estar **"local_da_imagem"** - indica o caminho para o arquivo.
- A indicação do caminho (PATH) pode ser absoluta ou relativa. No caso de ser relativa deve-se considerar o diretório corrente (diretório onde o documento HTML
- em questão, está).
- Como padrão as imagens mais comumente reconhecidas pelos browsers são as do formato GIF e JPG.
- **HEIGTH=n WIDTH=n** indicam a altura e largura que a imagem deve assumir (n em pixels), caso não sejam escritos a imagem assumirá seu tamanho original.

223

Comentários em HTML

<!-- comentários -->

- Os comentários em HTML são feitos pela seqüência de caracteres "<!--" para demarcar o início e "-->" para demarcar o fim.
- Todos os elementos que estiverem entre esse TAG serão ignorados pelo browser e não serão mostrados na tela.

224

FORMS HTML

- Os formulários HTML são uma de o HTML permitir que os usuários entrem com determinados dados e os mesmos sejam enviados para o local determinado pelo FORM.
- Deve-se notar que o HTML **não** prevê como esses dados serão manipulados. O HTML apenas fornece os meios do usuário entrar com os dados.

<FORM atributos> </FORM>

- Delimitam a existência de formulários. Dentro desse FORM devem existir outros TAG que permitam a inserção de dados.
- ACTION="url" - indica para onde os dados informados deverão ser enviados.

225

Atributos do FORM

- NAME="nome" - especifica um nome para o FORM.
- METHOD="tipo_envio" - especifica como esses dados serão enviados. Os métodos válidos são POST e GET.
  - **GET** - Por esse método os dados serão enviados juntamente com a URL, método default.

    ex: www.empresa.com.br/recebe.htm?v_cidade=curitiba&v_pais=Brasil
  - **POST** - Por esse método os dados serão enviados de maneira não visível pelo usuário (junto com o corpo do documento enviado para o servidor).
- Alguns servidores WWW têm limitações quanto ao tamanho da URL,portanto, se a quantidade de dados a ser enviada for muito grande, pelo método GET, esses servidores podem cortar parte dos dados.
- O método GET auxilia na depuração de programas, mas após essa fase é preferível optar pelo método POST.

226

TAG de FORMS

<INPUT atributos>

- O TAG input permite várias maneira de entrada de dados.
- atributos:
- TYPE="tipo" - especifica o tipo de entrada dos dados.

TYPE="TEXT"	Entrada tipo texto (tipo default)
TYPE="CHECKBOX"	Botão marcado/desmarcado
TYPE="RADIO"	Botão de marcado/desmarcado. Usado com outros, com o mesmo atributo nome (NAME), age como seleção "um de muitos".
TYPE="SUBMIT"	Botão que envia os dados do formulário corrente para a URL definida pelo atributo ACTION do TAG FORM <FORM ACTION="URL" >.
TYPE="RESET"	Causa o "reset" para a condição inicial (valores default) da página
TYPE="HIDDEN"	Representa um campo escondido. O usuário não pode interagir com este campo. O atributo VALUE especifica o valor do campo. O atributo NAME e VALUE são requeridos.
TYPE="PASSWORD"	Um campo assim definido não são ecoados na tela os valores digitados pelo usuário.

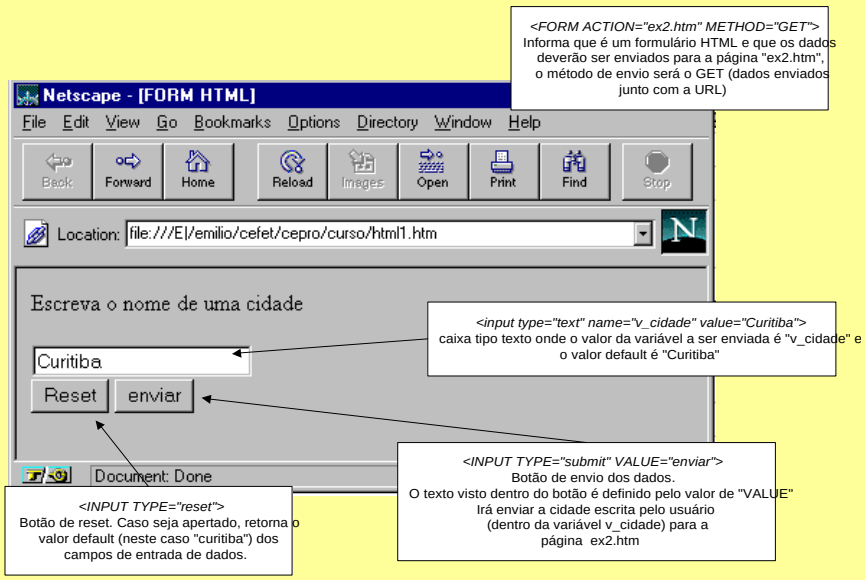
227

atributos <INPUT> (continuação)

NAME=	indica o nome da variável que irá ser associado com o dado definido por TYPE	
VALUE=	valor default.	Para o tipo "TEXT" exibe na área destinada para a entrada de texto, o valor especificado. Para o tipo "CHECKBOX" o valor indicado é enviado como valor da variável, caso o botão esteja marcado. Para o tipo "RESET" e "SUBMIT" o valor é o texto mostrado dentro do botão
CHECKED	Marca o botão caso o tipo seja "CHECKBOX"	
SIZE=	Tamanho da área de texto, mostrado para o usuário no tipo "TEXT"	obs deve-se notar que o tamanho (visualizado pelo usuário) das letras é variável.
MAXLEGTH=	máximo número de caracteres que podem ser digitados pelo usuário em um tipo "TEXT"	

228

Exemplo: <INPUT TYPE=TEXT ...>



<FORM ACTION="ex2.htm" METHOD="GET">
Informa que é um formulário HTML e que os dados deverão ser enviados para a página "ex2.htm", o método de envio será o GET (dados enviados junto com a URL)

<input type="text" name="v_cidade" value="Curitiba">
caixa tipo texto onde o valor da variável a ser enviada é "v_cidade" e o valor default é "Curitiba"

<INPUT TYPE="submit" VALUE="enviar">
Botão de envio dos dados.
O texto visto dentro do botão é definido pelo valor de "VALUE"
Irá enviar a cidade escrita pelo usuário (dentro da variável v_cidade) para a página ex2.htm

<INPUT TYPE="reset">
Botão de reset. Caso seja apertado, retorna o valor default (neste caso "curitiba") dos campos de entrada de dados.

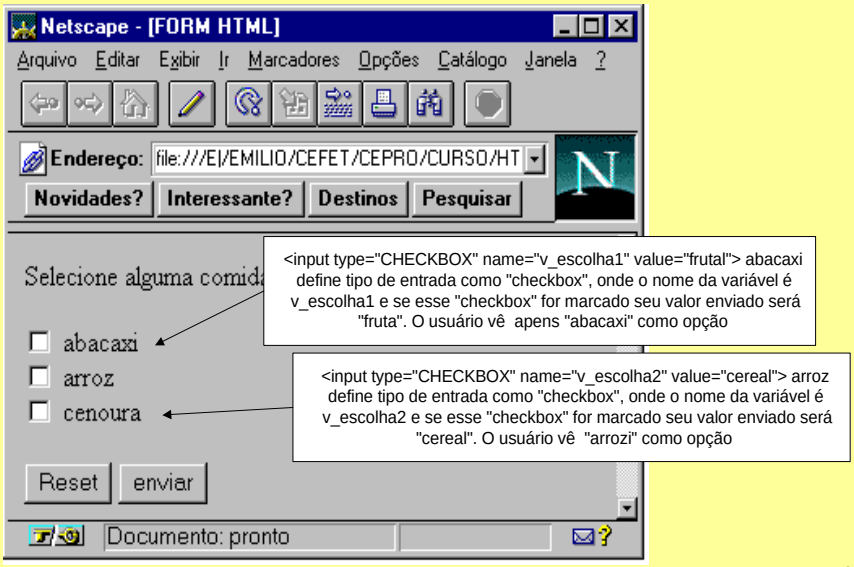
229

Código: <INPUT TYPE=TEXT ...>

```
<html>
  <body>
     <form action="htmltext.php" method="post">
          <br><br>
          Escreva o nome de uma cidade
          <br><br><br>
          <input type="text" name="p_cidade" size="20" maxlength="20">
          <input type=”hidden" name="p_cidade_velha" value=“Curitiba">
          <br><br><br>
          <input type="reset" value="Reset" >
          <input type="submit" value="Enviar" name="p_operacao">
     </form>
   </body>
<html>
```

230

Exemplo: <INPUT TYPE="checkbox" ... >



231

Código: <INPUT TYPE="checkbox" ... >

```
<html>
 <body>
  <form action="htmlcheckbox.php" method="post">
    <br><br>
    Selecione alguma comida
    <br><br><br>
    <input type="checkbox" name="p_escolha1" value="fruta1" >
    Abacaxi
    <input type="checkbox" name="p_escolha2" value="fruta2" >
    Arroz
    <input type="checkbox" name="p_escolha3" value="fruta3" >
    Cenoura
    <br><br><br>
    <input type="reset" value="Reset" >
    <input type="submit" value="Enviar" name="p_operacao">
  </form>
 </body>
<html>
```

232

<SELECT atributos> </SELECT>

- O TAG SELECT permite a seleção de um ou mais itens a partir de uma lista de opções.
- atributos:
- name=“nome” - identifica o valor da variável, dentro da qual as opções escolhidas serão enviadas.
- multiple - permite que um ou mais opções sejam selecionadas (em alguns browsers a multipla seleção é feita, apertando-se a tecla “ctrl” e selecionando a opção com o mouse)
- size=n - informa quantas opções serão vistas pelo usuário ao mesmo tempo, sem que haja necessidade de mover a barra de rolagem.

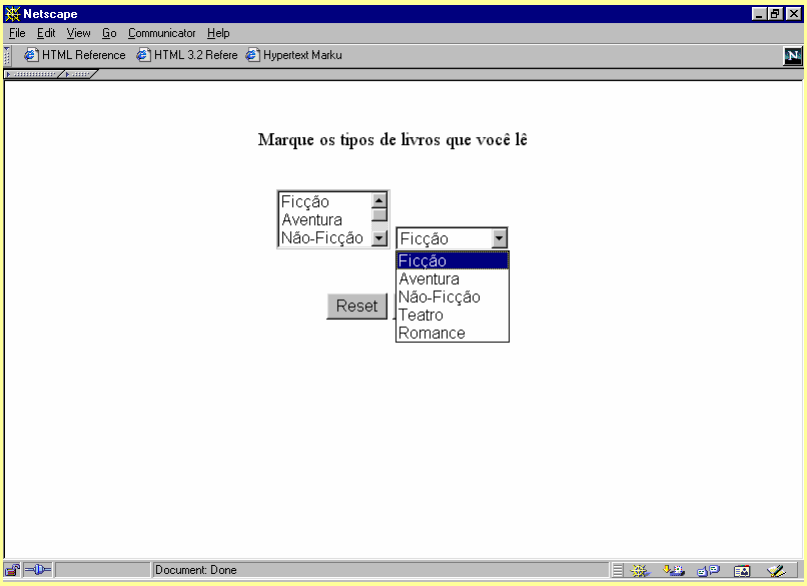
233

<OPTION atributos> valor_visível

- Esse TAG é valido somente entre o TAG SELECT. Serve para formar cada uma das opções que o usuário pode escolher. O “valor_visível” é o valor que o usuário irá ver na tela.
- Atributos:
- VALUE=“valor_enviar” - indica o valor a ser enviado caso a opção seja escolhida. Caso esse atributo não seja especificado, o valor enviado será o valor visto pelo usuário (“valor_visivel”)
- SELECTED - indica que esta opção é selecionada por default.

234

<SELECT...> e <OPTION ...>



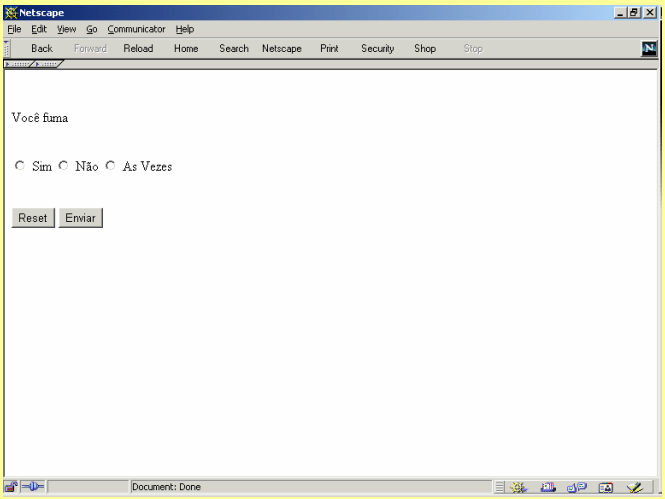
235

Código: <SELECT...> e <OPTION ...>

```
<html>
  <body>
      <center>
   <form action="htmlselect.php" method="post">
    <br><br>
    Marque os tipos de livros que você lê
    <br><br><br>
      <select name="p_livro" size=3 multiple>
            <option value="FC"> Ficção
            <option value="AV"> Aventura
            <option value="NF"> Não-Ficção
            <option value="TE"> Teatro
            <option value="RO"> Romance
      </select>
      <select name="p_livro">
            <option value="FC"`selected> Ficção
            <option value="AV"> Aventura
            <option value="NF"> Não-Ficção
            <option value="TE"> Teatro
            <option value="RO"> Romance
      </select>
    <br><br><br>
    <input type="reset" value="Reset" >
    <input type="submit" value="Enviar" name="p_operacao">
   </form>
  </body>
<html
```

236

Exemplo: <INPUT TYPE="radio" ... >



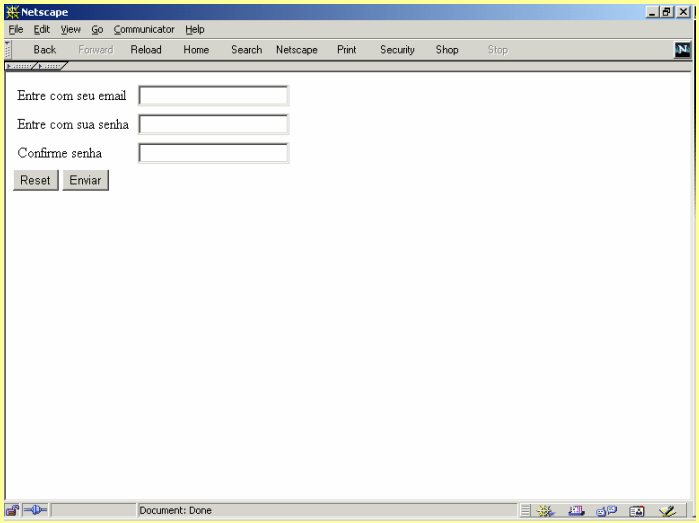
237

Código: <INPUT TYPE="radio" ... >

```
<html>
 <body>
  <form action="htmlradio.php" method="post">
    <br><br>
    Você fuma
    <br><br><br>
    <input type="radio" name="p_fuma" value="TRUE"> Sim
    <input type="radio" name="p_fuma" value="FALSE"> Não
    <input type="radio" name="p_fuma" value="TRUEFALSE"> As Vezes
    <br><br><br>
    <input type="reset" value="Reset" >
    <input type="submit" value="Enviar" name="p_operacao">
  </form>
 </body>
<html>
```

238

Exemplo: <INPUT TYPE="password" ... >



239

Código: <INPUT TYPE="password">

```
<html>
 <body>
  <form action="htmlpassword.php" method="post">
   <table>
    <tr>
      <td align="rigth">  Entre com seu email </td>
      <td align="rigth">
        <input type="text" name="p_login" size=20 maxlength=20 >
      </td>
   </tr> <tr>
      <td align="rigth"> Entre com sua senha </td>
      <td align="rigth">
        <input type="password" name="p_senha1" size=20 maxlength=20 >
      </td>
   </tr> <tr>
      <td align="rigth"> Confirme senha </td>
      <td align="rigth">
        <input type="password" name="p_senha2" size=20 maxlength=20 >
      </td>
   </table>
   <input type="reset" value="Reset" >
   <input type="submit" value="Enviar" name="p_operacao">
  </form>
 </body>
<html>
```

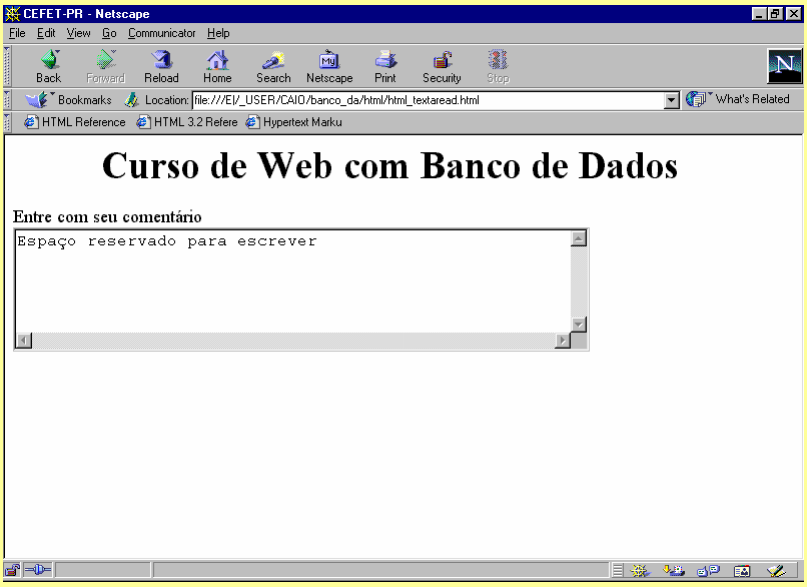
240

<TEXTAREA atributos> texto <TEXTAREA>

- Permite a inclusão de uma área de texto para a inserção de dados.
- O texto que estiver entre esses TAG será escrito dentro da área de texto, como valor default. Esse texto pode ser editado pelo usuário.
- atributos:
  - NAME=“nome” - identifica o valor da variável, dentro da qual o texto digitado irá ser enviado.
  - COLS=n - indica quantas colunas (dígitos) poderão aparecer na área de texto sem a necessidade de se mover a barra de rolagem
  - ROWS=n - indica o número de linhas visíveis sem a necessidade de mover a barra de rolagem.

241

exemplo : <TEXTAREA... > ...



242

Código <TEXTAREA>

```
<HTML>
    <HEAD> <TITLE> CEFET-PR </TITLE> <HEAD>
    <BODY bgcolor=white text=black  link=red vlink=red>
      <center>
      <H1> Curso de Web com Banco de Dados </H1>
          </center>
          <form>
                    Entre com seu comentário <BR>
                    <textarea name="p_comentario" cols=55 rows=5">
Espaço reservado para escrever
                    </textarea>
          </form>
    </BODY>
</HTML>
```

243

Frames

- Quadros ou divisões físicas na página.
- Uma página com frames apresenta dois ou mais arquivos HTML, carregadas uma em cada quadro.

<FRAMESET> </FRAMESET>

- Os atributos COLS e ROWS mutuamente excludentes.
- COLS - divide a janela em quadros um ao lado da outro.
  - COLS="25%,75%"
  - divide a página em dois quadros laterais, o primeiro com um 🕐 da largura da janela do navegador o eoutro com 🕒 da largura.
- ROWS - divide a janela um em cima do outro.
  - ROWS= "18%,200,*"
  - divide a página em três quadros verticais: com com 18% da altura outro com 200 pixels de altura e outro com que sobrar(*).

244

Atributos do <FRAMESET>

Caso se defina as janelas com porcentagem as proporções se mantem as mesmas, assim que as janelas são redimensionadas.

Caso se defina as janelas com pixels ela não é alterada mesmo depois de redimensionar a janela.

Dentro de </FRAMESET> insere-se os TAGS **<FRAME>** com atributo SRC, à página que deverá ser carregada naquela FRAME.

SRC="arquivo"

- Página que deverá ser carregada naquele FRAME

245

Atributos <FRAMESET>

- Cada FRAME pode ter um atributo **NAME**, que serve de alvo para as páginas a serem abertas.
- name="LISTA"
- noresize - Não permite que seja alterado o tamanho dos FRAMES.
- marginwidth=n - Delimita uma margem lateral para o conteúdo do FRAME.
- marginheight=n - Delimita uma margem vertical para o conteúdo do FRAME.
- scrolling="YES"|"NO"|"AUTO" - apresenta ou não barra de rolagem

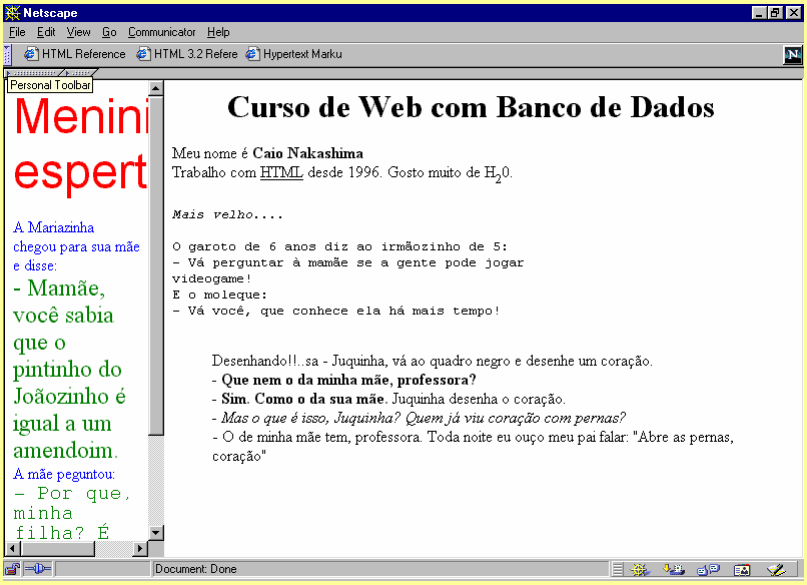
246

Frame Vertical e Horizontal

```
<html>
    <frameset cols="20%,*" border=0>
            <frame src="html_exercicio4.html" name="esquerda">
            <frame src="html_exercicio2.html" name="direita">
    </frameset>
</html>


<html>
    <frameset rows="20%,*" border=0>
            <frame src="html_exercicio4.html" name="cima">
            <frame src="html_exercicio2.html" name="baixo">
    </frameset>
</html>
```

247

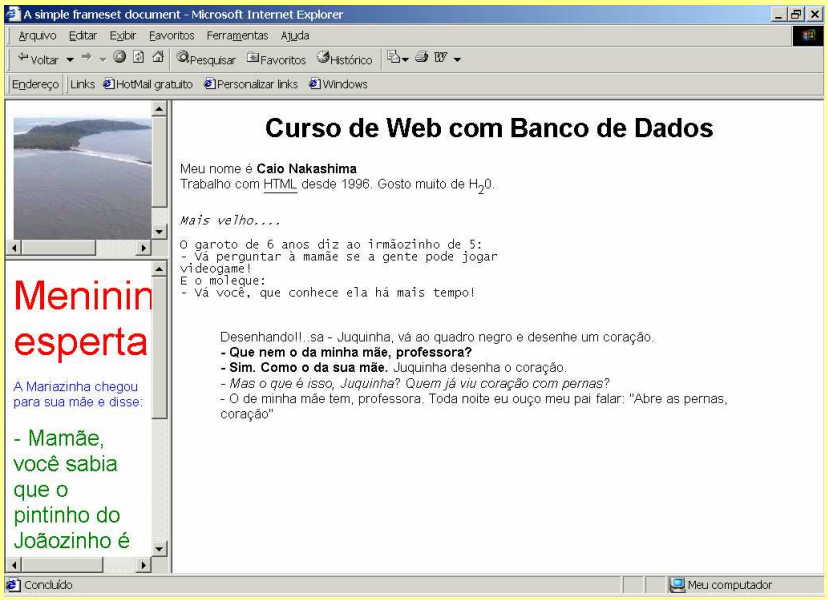


248

Frame Composto

```
<HTML>
<HEAD>
<TITLE>A simple frameset document</TITLE>
</HEAD>
<FRAMESET cols="20%, 80%">
  <FRAMESET rows="100, 200">
     <FRAME src="10-primeira-ilha.jpg">
      <FRAME src="html_exercicio4.html">
  </FRAMESET>
     <FRAME src="html_exercicio2.html">
  <NOFRAMES>
     <P>This frameset document contains:
     <UL>
       <LI><A href="html_exercicio4">Some neat contents</A>
       <LI><IMG src="10-primeira-ilha.jpg" alt="A neat image">
       <LI><A href="html_exercicio2.html">Some other neat contents</A>
     </UL>
  </NOFRAMES>
</FRAMESET>
</HTML>
```

249

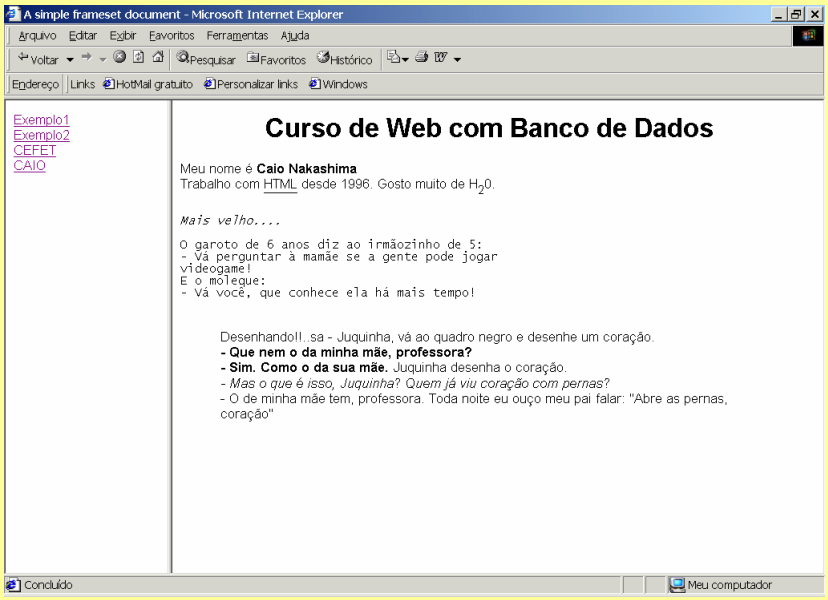


250

Frame com alvo

```
<html>
    <a href="html_exercicio1.html" target="meio"> Exemplo1 </a>
    <br>
    <a href="html_exercicio2.html" target="meio"> Exemplo2 </a>
    <br>
    <a href="http://www.cefetpr.br" target="_blank"> CEFET </a>
    <br>
    <a href="http://www.dainf.cefetpr.br/~nakashi" target="_top"> CAIO</a>

</html>
```

JavaScript

253

Introdução

- Criada pela SUN MicroSystem e Netscape.
- Necessidade de um browser inteligente, e uma linguagem de programação distribuída.
- Os scripts escritos com JavaScript podem ser colocados dentro das suas páginas HTML.
- Com JavaScript tem-se muitíssimas possibilidades de melhorar a sua página HTML com elementos interessantes.
- Pode-se de responder muito facilmente a eventos iniciados pelo usuário.
- A documentação esta publicada pela Netscape em http://home.netscape.com.

254

Introdução (cont.)

- Permite formular aplicações Internet no cliente e no servidor.
- JavaScript é uma linguagem oriunda de Java. Porém contem regras menos restritas e menos complexas.
- Exemplo: Mostrar o valor de um campo de entrada de texto dentro de um formulário que pertence a um documento HTML:

```
<html>
  <form>
   Digite nome
   <input type="TEXT" name="NOME"
   onChange="alert('O nome digitado foi '+this.value)">
  </form>
</html>
```

255

Diferença entre Java e JavaScript

- São duas técnicas diferentes de programação na Internet. Java é uma linguagem de programação.
- JavaScript é uma linguagem de scripting (tal como diz o nome).
- A diferença é que se pode criar programas reais com Java.
- Os autores de JavaScript não têm que se importar muito com programação.
- JavaScript é muito mais uma extensão do HTML do que uma linguagem de computador separada.
- Não é uma definição "oficial" mas acho que ela torna mais compreensível a diferença entre Java e JavaScript.

256

Exemplo - Write

```
<html>
    <head>
    Outra Forma de Chamar JavaScript!
    </head>
<body>
    <br>
    Este é um documento HTML normal.
    <br>
      <script language="JavaScript">
        document.write("Este é um JavaScript!")
      </script>
    <br>
    De volta ao HTML.
</body>
</html>
```

257

Manipulação de Eventos

Evento	Descrição
onBlur	Ocorre quando o foco sai de um campo
onClick	Ocorre quando um objeto é clicado
onChange	Ocorre quando um campo é alterado
onFocus	Ocorre quando o foco entra em um campo
onLoad	Ocorre quando um objeto é carregado
onUnload	Ocorre quando um objeto é fechado
onMouseOver	Ocorre quando o ponteiro do mouse passa sobre um objeto
onMouseOut	Ocorre quando o ponteiro do mouse sai de cima de um objeto
onSubmit	Ocorre quando um formulário é submetido.

258

```
<HTML>
<HEAD>
<TITLE>Java Script</TITLE>
<SCRIPT LANGUAGE="JavaScript">
     function donteventme( str ) {// Manipulador generico
     alert("Falei para não me " + str);
}
<!-- end script -->
</SCRIPT>
</HEAD>
     <BODY>
     <FORM METHOD="post" ACTION="mailto:nakashi@cefetpr.br">
     <BR>Não<INPUT TYPE="checkbox" NAME="meucheck" VALUE="HA!"
      onClick="donteventme('pressionar')">
     <SELECT NAME="mysel" onChange="donteventme('mudar')">
          <OPTION SELECTED>Nada</OPTION>
          <OPTION>Não eu</OPTION>
          <OPTION>De jeito nenhum</OPTION>
     </SELECT>
     </FORM>
</BODY>
</HTML>
```

259

Exemplo - Button

```
<html>
<head>
  <script language="JavaScript">
    function pushbutton() {
      alert("Alo!");
  }
 </script>
</head>
<body>
<form>
  <input type="button" name="Button1" value="Aperte-me" onClick="pushbutton()">
  </form>
</body>
</html>
```

260

onClick - onBlur - onFocus

- Evento em que o usuário clica o botão do mouse em um link ou em alguns elementos do formulário (onClick) ou quando o foco sai do elemento do formulário (onBlur).

```
<html>
    <form>
        Digite nome
        <input type="TEXT" name="NOME"
        onChange="alert('O nome digitado foi '+this.value)">
        <br>
        Digite rua
        <input type="TEXT" name="rua"
        onBlur="alert('A rua digitada foi '+this.value)">
        <br>
        Digite cidade
        <input type="TEXT" name=”cidade"
        onLoad="alert('Campo não obrigatório">
    </form>
</html>
```

261

onMouseOver

- Evento ativado com o mouse passa por cima de um elemento.

```
<html>
    <head>
    <script language="JavaScript">
     function hello() {
       alert("Alo!");
     }
    </script>
</head>
<body>
  <a href="html_exercicio1.html" onMouseOver="window.status='html exercicio 1';
    return true">
  Exemplo 1
  </a><br>
  <a href="html_exercicio2.html" onMouseOver="window.status='exemplo 2'; return
    true">
  Exemplo 2
  </a><br>
  <a href="" onMouseOver="hello()">link</a>
</body>
</html>
```

262

Elementos de um Browser

- **window** corresponde à janela. É o objeto de mais alto nível.
- **frame** corresponde às divisões de uma janela.
- **navigator** corresponde ao browser.
- **document** corresponde ao documento corrente.
- **location** corresponde ao URL corrente.
- **history** corresponde ao às URL que foi visitado.
- **forms** corresponde a um formulário inserido na página.
- **links** corresponde aos links que uma página contem.
- **anchors** corresponde a um pedaço de texto no mesmo documento
- **images** corresponde às imagens que são carregados no documento.

263

JavaScript em Páginas WEB

- A linguagem pode ser adicionada em uma página HTML de dois modos:
- Implementando o script dentro da própria página:

```
<script language="JavaScript">
   comandos JavaScript
</script>
```

- Chamando um arquivo texto com extensão .JS

```
<script src="endereco do arquivo/nomedoarquivo">
</script>
```

- Os comandos JavaScript ficam ocultos.

264

Operadores e Variáveis

- Não provê uma estrutura formal para declaração de constantes, fazendo com que qualquer identificador se comporte como variável.
- O próprio projetista deve preocupar-se em não sobrepor os valores das variáveis que deseja que se comportem como constantes.
  - Toda variável deve começar com uma letra ou um (“_”)
  - Caracteres subseqüentes devem ser letras ou números
  - Não deve conter espaço em branco ou caracteres especiais
  - Não deve ser uma palavra reservada
  - JavaScript é sensível ao caso (diferencia maiúscula de minúsculas)

265

Exemplo - Tipos de Variáveis

```
<html>
   <head>
        <script language="JavaScript">
               // Variavel Numérica
               total = 265;
               Total = 25.6;
               // Variavel Texto
               _outro = "valor";
               // Variavel Booleanas
               logico = true;   // true = 1
               fumante = false;  // false = 0
               // Variaveis Inválidas
               while = 0;              // Nome reservado
               12are = 9;              // Inicia com numero
        </script>
   </head>
   <body>

   </body>
</html>
```

266

Escopo de Variáveis

- Global: usada em qualquer local do script.
- Local: só pode ser usada dentro da função na qual foi declarada.
- Sintaxe:

```
var variável = valor;  // variável local
variável = valor; // variável global
```

- Declarar as variáveis globais no início do seu script, deixando o código melhor organizado.

267

```
<html>
  <head>
      <script language="JavaScript">
            // Atribução
            x = 145;
            curso = "WEB com Banco de Dados";
            fumante = false;
            // Operadores aritméticos
            x=10+5;  // valor de x é 15
            y=20-3;  // valor de y é 17
            z=120*2; // valor de z é 240
            w=30/2;  // valor de w é 15
            z=7%5;  // valor de z é 2
            // Operadores Bitwise
            ob1 = 15 & 9  // valor de ob1=9   1111&1001 = 1001
            ob2 = 15 | 9  // valor de ob2=15  1111|1001 = 1111
            ob3 = 15 ^ 9  // valor de ob3=6   1111^1001 = 0110
      </script>
  </head>
  <body>

  </body>
</html>
```

268

Operador

```
<html>
    <head>
      <script language="JavaScript">
                    function Resultado()
                    {
                              var op1 = document.operacao.valor1.value
                              op2 = document.operacao.valor2.value
                              op3=op1*op2
                    alert("Resposta="+op3)
                    }
       </script>
    </head>
    <body>
          <form name="operacao">
                    Primeiro Valor:
                    <input type="text" name="valor1" size=5><br>
                    Segundo Valor:
                    <input type="text" name="valor2" size=5><br>
                    <input type="button" onClick="Resultado()" name="Botao"
                              value="Resultado">
          </form>
    </body>
</html>
```


Exercício

1. Elabore uma página HTML com javaScript para testar os seguintes operadores:

    // Operadores aritméticos

    - subtração

    * multiplicação

    / divisão

    % resto

    // Operadores bitwise

    & e

    | ou

    ^ ou exclusivo

    >> deslocamento à direita

    << deslocamento à esquerda

Operadores Bitwise

- AND (“&”)
- 0&0 = 0
- 0&1 = 0
- 1&0 = 0
- 1&1 = 1

- OR (“|”)
- 0|0 = 0
- 0|1 = 1
- 1|0 = 1
- 1|1 = 1

- XOR (“^”)
- 0^0 = 0
- 0^1 = 1
- 1^0 = 1
- 1^1 = 0

- Left Shift (<<) - Desloca bits à esquerda.
  - Exemplo 9 << 1 = 18 (1001 << 1 = 10010)
- Right Shift (>>) - Desloca bits à direita.
  - Exemplo 9 >> 1 = 4 (1001 >> 1 = 100)

271

Operadores Lógicos/Relacionais

- Operador (E) expressão1 && expressão2
- Operador (OU) expressão1 || expressão2
- Operador (Não) ! expressão

- Operador (igual a) op1 == op2
- Operador (maior) op1 > op2
- Operador (menor) op1 < op2
- Operador (maior ou igual a) op1 >= op2
- Operador (menor ou igual a) op1 <= op2
- Operador (diferente de) op1 != op2

272

Operadores Short-Cut

Modo Normal	Short-Cut
x = x + y	x += y
x = x - y	x -= y
x = x * y	x *= y
x = x / y	x /= y
x = x & y	x &= y
x = x ^ y	x ^= y
x = x | y	x |= y
x = x << y	x <<= y
x = x >> y	x >>= y
x = x +1	x ++
x = x - 1	x--

273

Operador de Concatenação

```
<html>
   <head>
       <script language="JavaScript">
              // Operadores de concatenação
              x="Curso no ";
              y="CEFET";
              local = x + y; // Resulta em "Curso no CEFET"
       </script>
   </head>
   <body>

   </body>
</html>
```

274

Estrutura de Decisão

Sintaxe:

```
if (condição)
{
   bloco
}
else
{
   bloco
}
```

```
<html>
      <head>
            <script language="JavaScript">
                  valor = 14;
                  if (valor==14)
                  {     resposta = true;
                        valor ++;
                  }
                  else
                        resposta = false;
            </script>
      </head>
      <body>

      </body>
</html>
```

275

Estrutura de Decisão (2)

Sintaxe:

variável = expressão ? valor1 : valor2;

```
<html>
      <head>
            <script language="JavaScript">
                  resposta = (valor==14)?(true):(false);
            </script>
      </head>
      <body>

      </body>
</html>
```

276

Estruturas de Repetição

Sintaxe:

for (inicio; condição; incremento)

{

bloco

}

```
<html>
	<head >
		<script language="JavaScript">
			function teste()
			{
				for (i=0;i<10;i+=2)
				{
				  contador +=1;
				  alert ("Entrei = "+i);
				}
			}
		</script>
	</head>
	<body onload=teste()>

	</body>
</html>
```

277

Estruturas de Repetição (2)

Sintaxe:

while (condição)

{

bloco

}

```
<html>
	<head >
		<script language="JavaScript">
			function teste()
			{
				i=0;
				while (i<10)
				{
				   alert ("Entrei = "+i);
				   i+=2;
				}
			}
		</script>
	</head>
	<body onload=teste()>

	</body>
</html>
```

278

Criando Funções

Sintaxe:

```
function nomedafunção (p1, p2, ..., pn)
{
   bloco
   return valor; // opcional
}
```

- Função é um conjunto de comandos que pode executar uma determinada ação ou retornar um valor.
- Somente é executada, se for chamada pelo código JavaScript ou por um evento.

```
<script language="JavaScript">
   nomedafunção (p1, p2, ..., pn)
</script>
```

279

```
<html>
    <head>
        <script language="JavaScript">
            function ano (x)
            { var resultado="";
              if (x % 2 == 0)
                  resultado = "Este ano é par";
              else
                  resultado = "Este ano é impar";
              return resultado;
            }
        </script>
    </head>
    <body>
        <form name="exemplo">
            Digite um ano :
            <input type=text size=4 name="p_ano">
            <input type=button
               value="Clique aqui"
             onClick="alert(ano(document.exemplo.p_ano.value))">
        </form>
    </body>
</html>
```

280

Outro Exemplo

```
<html>
    <head>
        <script language="JavaScript">
            function mensagem()
            {     alert("Exemplo de uma função chamada por link")
            }
        </script>
    </head>
    <body>
        <a href="JavaScript:mensagem()"> Clique aqui </a>
    </body>
</html>
```

281

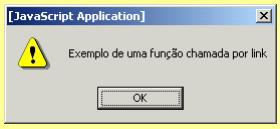
Exercício

- Criar um arquivo que contenha JavaScript para verificar se um determinado valor de um campo de um formulário é de ano bissexto ou não.
- Criar um arquivo que contenha JavaScript para imprimir o valor do fatorial de um determinado valor de um campo de formulário.
- Criar um arquivo que contenha JavaScript para imprimir todos os números primos de um determinado valor de um campo de formulário.
- Criar um arquivo que compara a string “CEFET-PR” e seu nome que vem de uma caixa de texto. Se a resposta for verdadeira aparece a mensagem (“senha correta”) senão aparece a mensagem (“senha incorreta”).

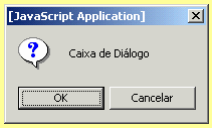
282

Caixa de Diálogo Simples

- alert(mensagem);
  - mensagem é a mensagem que irá aparecer na caixa de diálogo.



- variável = confirm(mensagem);
  - Retorna TRUE para OK e FALSE para Cancelar.



283

Caixa de Diálogo de Entrada

- variável = prompt("mensagem","valor padrão")
  - Utilizado quando a leitura de dados for simples e não justificar a criação de uma página HTML mais complexa.

```
<html>
  <head>
    <script language="JavaScript">
        function Teste()
        {
           aux=prompt("Entre com seu nome","Caio");
           alert(aux);
        }
    </script>
  </head>
    <body>
        <form>
            <input type="BUTTON" name=Nome value="Clique aqui"
             onclick="Teste()">
        </form>
    </body>
</html>
```

284

Escrevendo para um Documento

- document.write(“mensagem”);
- document.write(“mensagem”);
  - Escreve texto no objeto document.
  - O objeto document recebe o texto e mostra na página Web.
  - Após a execução de document.write() todas as variáveis em memória são reinicializados (perdidos).
- document.open();
  - abre o fluxo para um novo documento
- document.clear();
  - limpa um documento antes de executar write.
  - Executado automaticamente quanto write é utilizado
- document.close();
  - fecha o fluxo de escrita, descarregando o buffer.

285

Exemplo

```
<html>
  <head>
    <script language="JavaScript">
        function Teste()
        {
           var aux=prompt("Entre com seu nome","Caio");
           document.open();
           document.write(aux);
           document.close();
        }
    </script>
  </head>
    <body>
        <form>
            <input type="BUTTON" name=Nome value="Clique aqui"
             onClick="Teste()">
        </form>
    </body>
</html>
```

286

Criando janelas

```
<html>
<head>
    <script language="JavaScript">
    function Janela() {
      msg=open("","DisplayWindow","toolbar=no,directories=no,menubar=no");
      msg.document.write("<HEAD><TITLE>E a&iacute;?</TITLE></HEAD>");
      msg.document.write("<CENTER><h1><B>Isso &eacute; muito
    legal!</B></h1></CENTER>");
    }
    </script>
</head>
<body>
    <form>
    <input type="button" name="Button1" value="Aperte-me" onClick="Janela()">
    </form>
</body>
</html>
```

287

Temporizadores

- **setTimeout()** permite executar uma outra função após um tempo preestabelecido. Utilizada para funções repetitivas, típicas de aplicações de animação.
- **clearTimeout()** cancela o temporizador ativado por setTimeout.

```
<html>
<head>
<script language="JavaScript">
    function entrada()
            {window.location="10-primeira-ilha.jpg";        }
</script>
</head>
<body onLoad="setTimeout('entrada()',10000);">
    Esta pagina será substituida em breve.
</body>
</html>
```

288

Texto Rolante

```
<html>
<head>
<script language="JavaScript">
    var scrtxt="Aqui vai a mensagem para os visitantes da sua pagina que "+
    "ficarao olhando por horas de pura fascinacao...";
    var lentxt=scrtxt.length;
    var width=100;
    var pos=1-width;
    function scroll() {
      pos++;
      var scroller="";
      if (pos==lentxt)  pos=1-width;
          if (pos<0) {
            for (var i=1; i<=Math.abs(pos); i++) { scroller=scroller+" ";}
                                scroller=scroller+scrtxt.substring(0,width-i+1); }
          else {       scroller=scroller+scrtxt.substring(pos,width+pos); }

          window.status=scroller;
      setTimeout("scroll()",150);
    }
</script>
</head>
<body onLoad="scroll();return true;">
Aqui est&aacute; a sua p&aacute;gina bacana!
</body>
</html>
```

289

Apresentando a Data do Sistema

```
<html>
    <body>
        Esta é uma página-HTML simples.
        <br>
        Últimas modificações:
        <script language="JavaScript">
          today = new Date()
          document.write("A hora agora é: ",today.getHours(),
                ":",today.getMinutes(),".")
          document.write("A data é: ", today.getDate(),"/",
                today.getMonth()+1,"/",today.getYear())
        </script>
    </body>
</html>
```

290

Número Aleatório

```
<html>
    <head>
    <script language="JavaScript">
            function RandomNumber() {
              today = new Date();
              num= Math.abs(Math.sin(today.getTime()));
              return num;
            }
    </script>
</head>
<body>
    <script language="JavaScript">
    <!--
      document.write("Este é um número aleatório:", RandomNumber());
    // -->
    </script>
</body>
</html>
```

291

Formulários

- Cada formulário criado em HTML é inserido num vetor que inicia sua contagem 0.
- Assim, uma página contendo dois formulários pode acessar o primeiro formulário pelo índice 0 e o segundo pelo índice 1
- Propriedades
  - name - nome do formulário
  - action - nome do arquivo a ser chamado
  - method - forma de passagem do parâmetro
  - submit() - chamada do outro arquivo.

292

Manipulação de Dados

- **parseInt(valor)**
  - converte valores string em valores numéricos inteiros.
- **parseFloat(valor)**
  - converte valores string em valores numéricos de ponto flutuante.
- **eval(“expressão”)**
  - processa e calcula a expressão descrita com parâmetro, armazenado o resultado em variável.

293

Exemplo

```
<html>
<head>
<script language="JavaScript">
    function compute()       {
            document.forms[0].elements[1].value=
              eval(document.forms[0].elements[0].value);            }
</script>
</head>
<body>
    <form>
    Entre com uma expressão:
    <input type="text" size=20 name="campo1"><BR>
    Resultado:
    <input type="text" size=20 name="campo2"><BR>
    <input type="button" value="Clique aqui" onClick="compute()">
    </form>
</body>
</html>
```

294

Manipulação de String

- As strings são criadas automaticamente quando uma variável recebe um valor literal.
- Essa variável pode então acessar propriedades e métodos diretamente associados ao objeto string:
  - Possuem tags correspondentes.
  - big(), blink(), bold(), fixed(), italics(), small(), sub(), sup()
  - charAt(índice): retorna o caracter localizado no indice.
  - indexOf(valor,[índice]): busca a primeira ocorrência de uma string.
  - lastIndexOf(valor,[índice]): busca a última ocorrência de uma string.
  - substring(início, tamanho): extrai um substring.
  - toLowerCase(), toUpperCase(): converte em minúscula/maiúscula.

295

Exemplo STRING

```
<html>
<head>
<script language="JavaScript">
    function converte()       {
            document.Empregado.p_outro.value=
            document.Empregado.p_nome.value.big();
}
</script>
</head>
<body>
    <form name="Empregado">
    Entre com o nome
    <input type="text" size=20 name="p_nome" onBlur="converte()"><BR>
    Outro nome
    <input type="text" size=20 name="p_outro"><BR>
    </form>
</body>
</html>
```

296

Exercício

- Criar um arquivo HTML que apresente um relógio na página.
- Criar um arquivo HTML contendo:
  - um objeto checkbox que ao ser checado chame uma função que verifique se o checkbox esta ativado ou não.
  - dois objetos Text, sendo um que ao ter seu conteúdo alterado, emita uma mensagem mostrando o seu atual conteúdo. E o outro que ao perder o foco, mude o seu conteúdo para “Olá”.
  - um link que ao cursor se mover sobre ele apareça a mensagem “Oi estou aqui” na barra de status.
  - Um objeto texto onde deve ter sempre textos em maiúsculo.

297

Math

- É um objeto que armazena propriedades e métodos especializados para realização de cálculos matemáticos.
- Propriedades
  - E 2.718281828459045
  - LN10 2,302585092994045684017991454684 4
  - LN2 0,693147180559945309417232121458 18
  - PI 3,141592653589793238462643383279 5
  - SQRT1_2 0,707106781186547524400844362104 85
  - SQRT2 1,414213562373095048801688724209 7
- Métodos
  - abs(x), acos(x), atan(x), cos(x), sin(x), tan(x)
  - ceil(x), floor(x), max(x,y), min(x,y)
  - exp(x), pow(x,y), log(x)
  - round(x), sqrt(x)

298

Date

- A data é armazenada como o número de milisegundos desde 01/01/1970 às 00:00:00.
- Não se pode trabalhar com datas anteriores a 1/1/70.
- Intervalo de dados:
  - seconds e minutes 0 a 59
  - hours 0 a 23
  - day 0 a 6 (dias da semana)
  - date 1 a 31 (dia do mês)
  - month 0 (janeiro) a 11 (dezembro)
  - year de 1900

299

Métodos de Date

- getDate() Retorna o dia do mês
- getDay() Retorna o dia da semana
- getHours() Retorna a hora
- getMinutes() Retorna os minutos
- getMonth() Retorna o mês
- getSeconds() Retorna os segundos
- getTime() Retorna o valor numérico da hora
- getYear() Retorna o ano
- parse(date) Retorna o número em milisegundos na data desde 1/1/70 00:00:00
- setDate(dia) Estabelece o dia do mês para uma data específica.
- setMinutes(min) Estabelece o os minutos.
- setMonth(mês) Estabelece o mês
- setSeconds(seg) Estabelece os segundos
- setTime(valor) Estabelece o o valor do objeto date.
- setYear(ano) Estabelece o ano

300

Janelas (window)

- Window se refere a própria janela (como self), se uma página WEB não utilizar frames, as demais hierarquias tornam-se desnecessárias.
  - Window: refere-se a própria janela
  - self: refere-se a própria janela
  - top: refere-se a janela raiz
  - parent refere-se a janela pai
- Propriedades
  - status="mensagem" : exibe a mensagem durante a carga de um documento via Web.
  - defaultStatus="mensagem": é a mensagem que está configurada para se exibida após a carga de um documento na WEB.

    Serve para mostrar uma mensagem na linha de status na parte inferior da janela do Navegador.

    Pode ser utilizado em qualquer momento do programa.

301

Window - Métodos

var=window.open("url","nome", "parâmetros")

- Abre uma nova janela no Browser.
  - **var:** é uma variável que contém o endereço da janela aberta
  - **url**: endereço da nova janela, se for omitido abre uma janela vazia.
  - **nome:** especifica um nome que será usado internamente como referência.
  - **parâmetros:** informa a característica da janela a ser criada.
    - **toolbar= (yes || no)** cria uma barra de ferramenta no navegador
    - **location = (yes || no)** determina se mostra a URL
    - **directories = (yes || no)** determina se mostra What´s new e outros botões
    - **status = (yes || no)** barra de status na janela
    - **menubar = (yes || no)** barra de menu
    - **srollbars = (yes || no)** scroolbars
    - **resizeable = (yes || no)** mudar o tamanho da janela
    - **copyhistory = (yes || no)** se a nova janela herda a history da janela inicial
    - **width=n height=n** tamanho da janela

302

window.close()

- Fecha a janela especificada, liberando os recursos de memória utilizados durante a sua abertura.

```
<html>
 <head>
  <script language="JavaScript">
     function Teste()
     {     var beta= window.open('','janela3',
                    'width=250,height=250,screenX=450,screenY=50');
           beta.document.write("<img src='10-primeira-ilha.jpg'><br>");
           beta.document.write("<a href='JavaScript:self.close()'>");
           beta.document.write("Fecha</a>");
           beta.document.close();       }
  </script>
 </head>    <body>       <form>
        <input type="BUTTON" name=Nome value="Clique aqui" onclick="Teste()">
     </form>
  </body>
</html>
```

303

document

- É um objeto pré-construído pelo navegador e contém informações sobre conteúdo de sua página web.
- As informações utilizadas para construir o objeto são retiradas das informações colocadas em forma de tags HTML.
- Propriedades:
  - document.bgColor // Cor de fundo da tela
  - document.fgColor // cor do texto
  - document.linkColor // cor do link
  - document.alinkColor // cor do link ativo
  - document.vlinkColor // cor do link visitado
  - document.lastModified // última atualização no documento
  - document.referrer //URL da página que continha um link para página atual.

304

history

- Contém os endereços das páginas previamente visitadas.
  - history.length // Retorna o número de páginas visitadas
  - history.back() // Retorna para página anterior
  - history.forward() // Avança após ter efetuado o retorno
  - history.go(N) // Avança ou retorna diretamente N páginas

305

document.lastModifed()

```
<html>
   <body>
          <h1>Página do Caio Nakashima</h1>
           <BR>
       <script language="JavaScript">
                    document.write("Esta pagina foi alterada em "+
                                   document.lastModified);
     </script>

   </body>
</html>
```

306

Exercício

- Criar um arquivo HTML com o recurso de JavaScript. Criar uma tabela que contenha os botões de 0 a 9 e as operações matemáticas e um visor em que operações o resultado e os números digitados.
- Criar um formulário com 3 objetos do tipo TEXT e adicionar recurso JavaScript que façam com que um Texto tenha seus caracteres transformados em Maiúsculo.
- Criar um arquivo HTML com os seguintes recursos de JavaScript.
  - Criar botões com as funções de avançar e voltar as páginas
  - Criar um botão que retorne em uma caixa de mensagem quantas páginas já foram visitadas.

307

PHP

308

PHP

- PHP é uma linguagem que permite criar sites WEB dinâmicos, possibilitando uma interação com o usuário através de formulários, parâmetros da URL e links.
- A diferença de PHP com relação a linguagens semelhantes a Javascript é que o código PHP é executado no servidor, sendo enviado para o cliente apenas html puro.
- Desta maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente.

309

PHP

- Isso pode ser útil quando o programa está lidando com senhas ou qualquer tipo de informação confidencial.
- O que diferencia PHP de um script CGI escrito em C ou Perl é que o código PHP fica embutido no próprio HTML, enquanto no outro caso é necessário que o script CGI gere todo o código HTML, ou leia de um outro arquivo.
- PHP também tem como uma das características mais importantes o suporte a um grande número de bancos de dados, como dBase, Interbase, mSQL, mySQL, Oracle, Sybase, PostgreSQL e vários outros.
- Construir uma página baseada em um banco de dados torna-se uma tarefa extremamente simples com PHP.

310

Personal Home Page Tools

- A linguagem PHP foi concebida durante o outono de 1994 por **Rasmus Lerdorf.**
- Em meados de 1995 o interpretador foi reescrito, e ganhou o nome de **PHP/FI**, o "FI" veio de um outro pacote escrito por Rasmus que interpretava dados de formulários HTML (**F**orm **I**nterpreter).
- Estima-se que em 1996 PHP/FI estava sendo usado por cerca de 15.000 *sites* pelo mundo, e em meados de 1997 esse número subiu para mais de 50.000.
- O lançamento do PHP4, ocorrido em 22/05/2000, trouxe muitas novidades aos programadores de PHP.

311

Sintaxe Básica

- O código PHP fica embutido no próprio HTML. O interpretador identifica quando um código é PHP pelas seguintes tags:

```
<?php
comandos
?>
<script language="php">
comandos
</script>
<?
comandos
?>
<%
comandos
%>
```

312

Separador de Instruções

- Entre cada instrução em PHP é preciso utilizar o ponto-e-vírgula, assim como em C, Perl e outras linguagens mais conhecidas. Na última instrução do bloco de script não é necessário o uso do ponto-e-vírgula, mas por questões estéticas recomenda-se o uso sempre.

```
<html>
<header>
<title> Primeiro Exemplo PHP </title>
</header>
    Pode-se Escrever em HTML
<?
    Echo “Assim com em PHP”;
    Print (“<BR> </HTML>”);
?>
```

313

Nome de Variáveis

- Toda variável em PHP tem seu nome composto pelo caracter $ e uma string, que deve iniciar por uma letra ou o caracter “_”.
- **PHP é case sensitive**, ou seja, as variáveis $cn e $CN são diferentes.
- Por isso é preciso ter muito cuidado ao definir os nomes das variáveis.
- É bom evitar os nomes em maiúsculas, pois como veremos mais adiante, o PHP já possui alguma variáveis pré-definidas cujos nomes são formados por letras maiúsculas.

314

Comentários

- **Comentários de uma linha:**
  - Marca como comentário até o final da linha ou até o final do bloco de código PHP o que vier antes.
  - Pode ser delimitado pelo caracter "#" ou por duas barras (//).

```
<?  echo "teste"; #isto é um teste  ?>
<?  echo "teste"; //este teste é similar ao anterior ?>
```

315

Comentários

- **Comentários de mais de uma linha:**
  - Tem como delimitadores os caracteres "/*" para o início do bloco e "*/" para o final do comentário.
  - Se o delimitador de final de código PHP (?>) estiver dentro de um comentário, não será reconhecido pelo interpretador.

```
<?
echo "teste"; /* Isto é um comentário com mais
de uma linha, mas não funciona corretamente ?>
*/

<?
  echo "teste"; /* Isto é um comentário com mais
  de uma linha que funciona corretamente
*/
?>
```

316

Imprimindo código HTML

- Um script php geralmente tem como resultado uma página html, ou algum outro texto.
- Para gerar esse resultado, deve ser utilizada uma das funções de impressão, echo e print.
- Para utilizá-las deve-se utilizar um dos seguintes formatos:

```
print(argumento);
echo (argumento1, argumento2, ... );
echo argumento;
```

317

Tipos de Variáveis

- **PHP suporta os seguintes tipos de dados:**
  1. Inteiro
  2. Ponto flutuante
  3. String
  4. Array
  5. Objeto
- PHP utiliza checagem de tipos dinâmica, ou seja, uma variável pode conter valores de diferentes tipos em diferentes momentos da execução do script.
- Por este motivo não é necessário **declarar o tipo** de uma variável para usá-la.
- O interpretador PHP decidirá qual o tipo daquela variável, verificando o conteúdo em tempo de execução.

318

Integer, Long

- Uma variável pode conter um valor inteiro com atribuições que sigam as seguintes sintaxes:

```
$cn = 1234;          # inteiro positivo na base decimal
$cn = -234; # inteiro negativo na base decimal
$cn = 0234;          # inteiro na base octal-simbolizado pelo 0
                     # equivale a 156 decimal
$cn = 0x34;# inteiro na base hexadecimal(simbolizado
        # pelo 0x) – equivale a 52 decimal.
```

- A diferença entre inteiros simples e long está no número de bytes utilizados para armazenar a variável.
- Como a escolha é feita pelo interpretador PHP de maneira transparente para o usuário, podemos afirmar que os tipos são iguais.

319

Double ou Float

- Uma variável pode ter um valor em ponto flutuante com atribuições que sigam as seguintes sintaxes:

```
$cn = 1.234;
$cn = 23e4;          # equivale a 230.000
```

320

String

- Strings podem ser atribuídas de duas maneiras:
- Utilizando aspas simples (') – Desta maneira, o valor da variável será exatamente o texto contido entre as aspas (com exceção de \\ e \' – ver tabela abaixo)
- Utilizando aspas duplas (") – Desta maneira, qualquer variável ou caracter de escape será expandido antes de ser atribuído.

```
<?
   $teste = ”Caio";
   $cn = '---$teste--\n';
   echo "$cn";
?>
```

A saída desse script será "---$teste--\n".

```
<?
   $teste = ”Caio";
   $cn = "---$teste---\n";
   echo "$cn";
?>
```

A saída desse script será "---Caio--" (com uma quebra de linha no final).

321

Caracteres de Escape

- A tabela seguinte lista os caracteres de escape:

Sintaxe	*Significado*
\n	Nova linha
\r	Retorno de carro (semelhante a \n)
\t	Tabulação horizontal
\\	A própria barra (\ )
\$	O símbolo $
\’	Aspa simples
\”	Aspa dupla

322

Arrays

- Arrays em PHP podem ser observados como mapeamentos ou como vetores indexados.
- Mais precisamente, um valor do tipo array é um dicionário onde os índices são as chaves de acesso.
- Vale ressaltar que os índices podem ser valores de qualquer tipo e não somente inteiros.
- Inclusive, se os índices forem todos inteiros, estes não precisam formar um intervalo contínuo.
- Como a checagem de tipos em PHP é dinâmica, valores de tipos diferentes podem ser usados como índices de array, assim como os valores mapeados também podem ser de diversos tipos.

323

Arrays

Exemplo:

```
<?
  $cor[1] = "vermelho";
  $cor[2] = "verde";
  $cor[3] = "azul";
  $cor["teste"] = 1;
?>
```

Equivalentemente, pode-se escrever:

```
<?
  $cor = array(1 => "vermelho", 2 => "verde", 3 => "azul", "teste" =>
    1);
?>
```

324

Listas

- As listas são utilizadas em PHP para realizar atribuições múltiplas.
- Através de listas é possível atribuir valores que estão num array para variáveis. Vejamos o exemplo:

  list($a, $b, $c) = array("a", "b", "c");
- O comando acima atribui valores às três variáveis simultaneamente.
- É bom notar que só são atribuídos às variáveis da lista os elementos do array que possuem índices inteiros e não negativos.
- No exemplo acima as três atribuições foram bem sucedidas porque ao inicializar um array sem especificar os índices eles passam a ser inteiros, a partir do zero.

325

Listas

- Um fator importante é que cada variável da lista possui um índice inteiro e ordinal, iniciando com zero, que serve para determinar qual valor será atribuído.
- No exemplo anterior temos $a com índice 0, $b com índice 1 e $c com índice 2.
- Outro exemplo:

  $arr = array(1=>"um",3=>"tres","a"=>"letraA",2=>"dois");

  list($a,$b,$c,$d) = $arr;
- Após a execução do código acima temos os seguintes valores:

  $a == null

  $b == "um"

  $c == "dois"

  $d == "tres"

326

Listas

- Observa-se que à variável $a não foi atribuído valor, pois no array não existe elemento com índice 0 (zero).
- Outro detalhe importante é que o valor "tres" foi atribuído à variável $d, e não a $b, pois seu índice é 3, o mesmo que $d na lista.
- Por fim, ve-se que o valor "letraA" não foi atribuído a elemento algum da lista pois seu índice não é inteiro.
- Os índices da lista servem apenas como referência ao interpretador PHP para realizar as atribuições, não podendo ser acessados de maneira alguma pelo programador.
- De maneira diferente do array, uma lista não pode ser atribuída a uma variável, servindo apenas para fazer múltiplas atribuições através de um array.

327

Objetos

- Um objeto pode ser inicializado utilizando o comando *new* para instanciar uma classe para uma variável.

```
Exemplo:
class teste {
    function nada() {
            echo "nada";
    }
}

$cns = new teste;
$cns -> nada();
```

328

Booleanos

- PHP não possui um tipo booleano, mas é capaz de avaliar expressões e retornar *true* ou *false*, através do tipo integer: é usado o valor 0 (zero) para representar o estado *false*, e qualquer valor diferente de zero (geralmente 1) para representar o estado *true*.

329

Coerção

- Quando ocorrem determinadas operações ("+", por exemplo) entre dois valores de tipos diferentes, o PHP converte o valor de um deles automaticamente (coerção).
- É interessante notar que se o operando for uma variável, seu valor não será alterado.
- O tipo para o qual os valores dos operandos serão convertidos é determinado da seguinte forma:
- Se um dos operandos for float, o outro será convertido para float, senão, se um deles for integer, o outro será convertido para integer.

```
$cn = "1";              // $cn é a string "1"
$cn = $cn + 1;          // $cn é o integer 2
$cn = $cn + 3.7;        // $cn é o double 5.7
$cn = 1 + 1.5           // $cn é o double 2.5
```

Nota-se, o PHP converte string para integer ou double mantendo o valor.

330

Coerção Exemplo

- O sistema utilizado pelo PHP para converter de *strings* para números é o seguinte:

A. É analisado o início da string. Se contiver um número, ele será avaliado. Senão, o valor será 0 (zero);

B. O número pode conter um sinal no início ("+" ou "-");

C. Se a string contiver um ponto em sua parte numérica a ser analisada, ele será considerado, e o valor obtido será double;

D. Se a string contiver um "e" ou "E" em sua parte numérica a ser analisada, o valor seguinte será considerado como expoente da base 10, e o valor obtido será double;

```
$cn = 1 + "10.5";          // $cn == 11.5
$cn = 1 + "-1.3e3";        // $cn == -1299
$cn = 1 + "teste10.5";     // $cn == 1
$cn = 1 + "10testes";      // $cn == 11
$cn = 1 + "  10testes";    // $cn == 11
$cn = 1 + "+  10testes";   // $cn == 1
```

331

Transformação explícita de tipos

- A sintaxe do *typecast* de PHP é semelhante ao C: basta escrever o tipo entre parenteses antes do valor

```
$cn = 15;                  // $cn é integer (15)
$cn = (double) $cn         // $cn é double (15.0)
$cn = 3.9                  // $cn é double (3.9)
$cn = (int) $cn            // $cn é integer (3)
                           // o valor decimal é truncado
```

- Os tipos de *cast* permitidos são:

(int), (integer)	Þ muda para integer;
(real), (double), (float)	Þ muda para float;
(string)	Þ muda para string;
(array)	Þ muda para array;
(object)	Þ muda para objeto.

332

Com a função settype

- A função settype converte uma variável para o tipo especificado, que pode ser "integer", "double", "string", "array" ou "object".

Exemplo:

```
$cn = 15;                  // $cn é integer
settype($cn,double)        // $cn é double
```

333

Operadores Aritméticos/String

- Só podem ser utilizados quando os operandos são números (integer ou float). Se forem de outro tipo, terão seus valores convertidos antes da realização da operação.

+	adição
-	subtração
*	multiplicação
/	divisão
%	módulo

- Só há um operador exclusivo para strings:

.	concatenação

334

Operadores de Atribuição

- Existe um operador básico de atribuição e diversos derivados. Sempre retornam o valor atribuído.
- No caso dos operadores derivados de atribuição, a operação é feita entre os dois operandos, sendo atribuído o resultado para o primeiro.
- A atribuição é sempre por valor, e não por referência.

=	atribuição simples
+=	atribuição com adição
-=	atribuição com subtração
*=	atribuição com multiplicação
/=	atribuição com divisão
%=	atribuição com módulo
.=	atribuição com concatenação

Exemplo

```
$a = 7;
$a += 2; // $a passa a conter o valor 9
```

335

Bit a Bit - Lógico

- Comparam dois números bit a bit.

&	"e" lógico
|	"ou" lógico
^	ou exclusivo
~	não (inversão)
<<	shift left
>>	shift right

- Utilizados para inteiros representando valores booleanos

and	"e" lógico
or	"ou" lógico
xor	ou exclusivo

!não (inversão)

&&	"e" lógico
||	"ou" lógico

- Existem dois operadores para "e" e para "ou porque eles têm diferentes posições na ordem de precedência.

336

Comparação

- As comparações são feitas entre os valores contidos nas variáveis, e não as referências.
- Sempre retornam um valor booleano.

==	igual a
!=	diferente de
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a

- Existe um operador de seleção que é ternário. Funciona assim:

  (expressao1)?(expressao2):(expressao3)

- o interpretador PHP avalia a primeira expressão. Se ela for verdadeira, a expressão retorna o valor de expressão2. Senão, retorna o valor de expressão3.

337

Incremento e Decremento

++ incremento
-- decremento

- Podem ser utilizados de duas formas: antes ou depois da variável.
- Quando utilizado antes, retorna o valor da variável antes de incrementá-la ou decrementá-la.
- Quando utilizado depois, retorna o valor da variável já incrementado ou decrementado.
- Exemplos:

```
$a = $b = 10;      // $a e $b recebem o valor 10
$c = $a++;         // $c recebe 10 e $a passa a ter 11
$d = ++$b;         // $d recebe 11, valor de $b já incrementado
```

338

Escopo e Visibilidade de Variáveis

- O escopo de uma variável é o contexto dentro o qual está definido.
- A maior parte todas as variáveis de PHP só têm um único escopo.
- Porém, dentro de funções definidas pelo usuário é introduzida uma extensão de função local.
- Qualquer variável usada dentro de uma função está por falta limitada à extensão de função local. Por exemplo:
-
```
$a = 1; /*escopo global */
Function Test () {
    echo $a; /* referencia a variavel local */
}
Test ();
```
- Resultado será nulo pois esta referenciando a uma variável local.

339

Escopo e Visibilidade de Variáveis

- Nota-se que este é um pouco diferente da linguagem C naquelas variáveis globais em C está automaticamente disponível a funções a menos que especificamente anulasse por uma definição local.
- Isto pode causar alguns problemas dentro que as pessoas podem mudar uma variável global inadvertidamente.
- Em PHP devem ser declaradas variáveis globais global dentro de uma função se eles forem ser usados naquela função. Um exemplo:

```
$a = 1;
$b = 2;
Function Sum () {
    global $a, $b;
    $b = $a + $b;
}
Sum ();
echo $b;
```

- Será impresso o valor “3”.

340

Escopo e Visibilidade de Variáveis

- Declarando $a e $b global dentro da função, todas as referências para qualquer variável recorrerão à versão global. Não há nenhum limite ao número de variáveis globais que podem ser manipuladas por uma função.
- Um segundo modo para ter acesso variáveis da extensão global é usar a ordem de $GLOBALS PHP-definida especial. O exemplo prévio pode ser rescrito como:

```
$a = 1;
$b = 2;

Function Sum () {
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Sum ();
echo $b;
```

341

Escopo e Visibilidade de Variáveis

- Um vetor (array) de $GLOBALS é um vetor associativo com o nome do ser variável global a chave e o conteúdo daquela variável que é o valor do elemento de ordem.
- Outra característica importante de escopo de uma variável é a variável estática. Uma variável estática só existe em uma extensão de função local, mas não perde seu valor quando programa execução deixa esta extensão. Considere o exemplo seguinte:

```
Function Test () {
    $a = 0;
    echo $a;
    $a++;
}
```

- Esta função é bastante inútil desde toda vez isto é chamado fixa $a a 0 e imprime " 0 .

342

Escopo e Visibilidade de Variáveis

- O $a++ que incrementa os saques variáveis desde então nenhum propósito assim que a função saia que a variável de $a desaparece. Para fazer uma função contando útil que não perderá de vista da conta atual, a variável de $a é declarada estático:

```
Function Test () {
    static $a = 0;
    echo $a;
    $a++;
}
```

- Agora, toda vez o Teste () função é chamada imprimirá o valor de $a e incrementará isto.

343

Variáveis Estáticas

- Variáveis estáticas também são essenciais quando são chamadas funções recursivas.
- Deve-se tomado cuidado ao escrever uma função recursiva porque é possível fazer isto indefinidamente.
- Você deve ter certeza você tem um modo adequado de terminar a recursividade.
- A função abaixo será executada até o count chegar a 10.

```
Function Test () {
    static $count = 0;
    $count++;
    echo $count;
    if ($count < 10) {
        Test ();
    }
    $count--;
}
```

344

Variáveis Variáveis

- O PHP tem um recurso conhecido como variáveis variáveis, que consiste em variáveis cujos nomes também são variáveis. Sua utilização é feita através do duplo cifrão ($$).

```
$a = "teste";
$$a = "Hiromi";
```

O exemplo acima e equivalente ao seguinte:

```
$a = "teste";
$teste = "Hiromi";
```

345

Variáveis enviadas pelo navegador

- Para interagir com a navegação feita pelo usuário, é necessário que o PHP possa enviar e receber informações para o software de navegação.
- A maneira de enviar informações, como já foi visto anteriormente, geralmente é através de um comando de impressão, como o *echo*.
- Para receber informações vindas do navegador através de um *link* ou um formulário html o PHP utiliza as informações enviadas através da URL.
- Por exemplo: se seu script php está localizado em "http://localhost/teste.php3" e você o chama com a url http://localhost/teste.php3?cn=teste", automaticamente o PHP criará uma variável com o nome $cn contendo a string "teste".
- Note que o conteúdo da variável está no formato urlencode. Os formulários html já enviam informações automaticamente nesse formato, e o PHP decodifica sem necessitar de tratamento pelo programador.

346

URLencode

- O formato urlencode é obtido substituindo os espaços pelo caracter "+" e todos os outros caracteres não alfa-numéricos (com exceção de "_") pelo caracter "%" seguido do código ASCII em hexadecimal.
- Por exemplo: o texto "Testando 1 2 3 !!" em urlencode fica "Testando+1+2+3+%21%21"
- O PHP possui duas funções para tratar com texto em urlencode. Seguem suas sintaxes:

  string **urlencode**(string texto);

  string **urldecode**(string texto);
- Essas funções servem respectivamente para codificar ou decodificar um texto passado como argumento. Para entender melhor o que é um argumento e como funciona uma função, leia o tópico "funções".

347

Utilizando Arrays

- Cada elemento de um formulário HTML submetido a um script PHP cria no ambiente do mesmo uma variável cujo nome é o mesmo nome do elemento.
- Por exemplo: um campo definido como:

```
<input type="text" name="endereco">
```

- ao ser submetido a um script PHP fará com que seja criada uma variável com o nome $endereco.
- Isto acontece de forma semelhante para cookies, como veremos mais adiante.
- Uma boa técnica de programação é utilizar a notação de arrays para nomes de cookies ou itens de um formulário html.

348

Utilizando Arrays

- Para um conjunto de checkboxes, por exemplo, podemos utilizar a seguinte notação:

```
<input type="checkbox" name="teste[]" value="valor1">opcao1
<input type="checkbox" name="teste[]" value="valor2">opcao2
<input type="checkbox" name="teste[]" value="valor3">opcao3
<input type="checkbox" name="teste[]" value="valor4">opcao4
<input type="checkbox" name="teste[]" value="valor5">opcao5
```

- Ao submeter o formulário, o script que recebe os valores submetidos terá uma variável chamada $teste contendo os valores marcados num array, com índices a partir de zero.

349

Utilizando Arrays

- Assim, se forem marcadas as opções 2, 3 e 5, poderemos fazer as seguintes afirmações:

```
$teste == array("valor2", "valor3", valor5");
$teste[0] == "valor2";
$teste[1] == "valor3";
$teste[2] == "valor5";
```

- O mesmo artifício pode ser utilizado com outros elementos de formulários e até com cookies.

350

Variáveis de ambiente

- O PHP possui diversas variáveis de ambiente, como a $PHP_SELF, por exemplo, que contém o nome e o path do próprio arquivo.
- Algumas outras contém informações sobre o navegador do usuário, o servidor http, a versão do PHP e diversas informações.
- Para ter uma listagem de todas as variáveis e constantes de ambiente e seus respectivos conteúdos, deve-se utilizar a função phpinfo().

```
<?
   echo phpinfo();
?>
```

351

Destruindo uma variável

- É possível desalocar uma variável se ela não for usada posteriormente através da função unset, que tem a seguinte assinatura:

```
int unset(mixed var);
```

- A função destrói a variável, ou seja, libera a memória ocupada por ela, fazendo com que ela deixe de existir.
- Se mais na frente for feita uma chamada á variável, será criada uma nova variável de mesmo nome e de conteúdo vazio, a não ser que a chamada seja pela função isset.
- Se a operação for bem sucedida, retorna true.

352

Verificando se uma variável possui um valor

- Existem dois tipos de teste que podem ser feitos para verificar se uma variável está setada: com a função isset e com a função empty.
- **A função isset**

  int **isset**(mixed var);
- E retorna true se a variável estiver setada (ainda que com uma string vazia ou o valor zero), e false em caso contrário.

- **A função empty**

  int **empty**(mixed var);
- E retorna true se a variável não contiver um valor (não estiver setada) ou possuir valor 0 (zero) ou uma string vazia. Caso contrário, retorna false.

353

Constantes pré-definidas

- O PHP possui algumas constantes pré-definidas, indicando a versão do PHP, o Sistema Operacional do servidor, o arquivo em execução, e diversas outras informações. Para ter acesso a todas as constantes pré-definidas, pode-se utilizar a função phpinfo(), que exibe uma tabela contendo todas as constantes pré-definidas, assim como configurações da máquina, sistema operacional, servidor http e versão do PHP instalada.
- Para definir constantes utiliza-se a função define. Uma vez definido, o valor de uma constante não poderá mais ser alterado. Uma constante só pode conter valores escalares, ou seja, não pode conter nem um array nem um objeto. A assinatura da função define é a seguinte:

  int **define**(string nome_da_constante, mixed valor);
- A função retorna true se for bem-sucedida. Veja um exemplo de sua utilização a seguir:

```
define ("pi", 3.1415926536);
$circunf = 2*pi*$raio;
```

354

Verificando o tipo de uma variável

- Função que retorna o tipo da variável é a gettype. Sua assinatura é a seguinte:

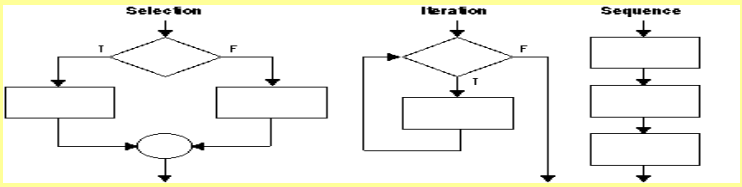
  string **gettype**(mixed var);
- A palavra "**mixed**" indica que a variável var pode ser de diversos tipos.
- A função **gettype** pode retornar as seguintes strings: "integer", "double", "string", "array", "object" e unknown type".
- Funções que testam o tipo da variável, são as funções is_int, is_integer, is_real, is_long, is_float, is_string, is_array e is_object. Todas têm o mesmo formato, seguindo modelo da assinatura a seguir:

  int **is_integer**(mixed var);
- Todas essas funções retornam true se a variável for daquele tipo, e false em caso contrário.

355

Estruturas de Controle

- Permite processar dados utilizado comando condicionais, iterativos e com controle de fluxo.
  - IF ... THEN ... ELSE ...
  - FOR ... LOOP
  - WHILE ... LOOP
  - EXIT WHEN
  - GOTO



356

Blocos

- Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função.
- Em comandos como if, for, while, switch e em declarações de funções blocos podem ser utilizados para permitir que um comando faça parte do contexto desejado.
- Blocos em PHP são delimitados pelos caracteres "{" e "}". A utilização dos delimitadores de bloco em uma parte qualquer do código não relacionada com os comandos citados ou funções não produzirá efeito algum, e será tratada normalmente pelo interpretador.

357

Blocos

- No exemplo abaixo, o comando2 sempre será executado:

```
if ($x == $y)
  comando1;
  comando2;
```

- Para que comando2 esteja relacionado ao if é preciso utilizar um bloco:

```
if ($x == $y){
  comando1;
  comando2;
}
```

358

Controle Condicional

- **IF-ELSE** permite executar uma seqüência de comados segundo uma ou mais condições.
- O mais trivial dos comandos condicionais é o if. Ele testa a condição e executa o comando indicado se o resultado for true (valor diferente de zero). Ele possui duas sintaxes:

```
if (expressão)
   comando;

if (expressão):
   comando;
   . . .
   comando;
endif;
```

- Para incluir mais de um comando no if da primeira sintaxe, é preciso utilizar um bloco, demarcado por chaves.

359

IF ... THEN ...

- O else é um complemento opcional para o if.
- Se utilizado, o comando será executado se a expressão retornar o valor false (zero). Suas duas sintaxes são:

```
if (expressão)
   comando;
else
   comando;

if (expressão):
   comando;
   . . .
   comando;
else:
   comando;
   . . .
   comando;
endif;
```

360

IF

- A seguir, temos um exemplo do comando if utilizado com else:

```
if ($a > $b)
  $maior = $a;
else
  $maior = $b;
```

- O exemplo acima coloca em $maior o maior valor entre $a e $b
- Em determinadas situações é necessário fazer mais de um teste, e executar condicionalmente diversos comandos ou blocos de comandos.

361

IF

- Para facilitar o entendimento de uma estrutura do tipo:

```
if (expressao1)
   comando1;
else
  if (expressao2)
    comando2;
  else
    if (expressao3)
      comando3;
    else
      comando4;
```

- sequencialmente, como no exemplo acima.
- Num mesmo if podem ser utilizados diversos elseif's, ficando essa utilização a critério do programador, que deve zelar pela legibilidade de seu script.

362

ELSEIF

- O comando elseif também pode ser utilizado com dois tipos de sintaxe.
- Em resumo, a sintaxe geral do comando if fica das seguintes maneiras:

```
if (expressao1)
  comando;
[ elseif (expressao2)
   comando; ]
[ else
   comando; ]
```

```
if (expressao1) :
  comando;
   . . .
  comando;
[ elseif (expressao2):
  comando;
   . . .
  comando; ]
[ else:
  comando;
   . . .
  comando; ]
endif;
```

363

Dica

```
IF condition1
  statement1;
ELSE
  IF condition2
    statement2;
  ELSE
    IF condition3
      statement3;
    END IF;
  END IF;
END IF;
```

```
IF condition1
  statement1;
ELSEIF condition2
  statement2;
ELSEIF condition3
  statement3;
END IF;
```

364

SWITCH

- O comando switch atua de maneira semelhante a uma série de comandos if na mesma expressão.
- Frequentemente o programador pode querer comparar uma variável com diversos valores, e executar um código diferente a depender de qual valor é igual ao da variável.
- Quando isso for necessário, deve-se usar o comando switch.
- O exemplo seguinte mostra dois trechos de código que fazem a mesma coisa, sendo que o primeiro utiliza uma série de if's e o segundo utiliza switch:

365

Exemplo switch

```
if ($i == 0)
   print "i é igual a zero";
elseif ($i == 1)
   print "i é igual a um";
elseif ($i == 2)
   print "i é igual a dois";

switch ($i) {
case 0:
     print "i é igual a zero";
     break;
case 1:
     print "i é igual a um";
     break;
case 2:
     print "i é igual a dois";
     break;
}
```

366

Switch

- É importante compreender o funcionamento do switch para não cometer enganos.
- O comando switch testa linha a linha os cases encontrados, e a partir do momento que encontra um valor igual ao da variável testada, passa a executar todos os comandos seguintes, mesmo os que fazem parte de outro teste, até o fim do bloco. por isso usa-se o comando break, quebrando o fluxo e fazendo com que o código seja executado da maneira desejada. Veremos mais sobre o break mais adiante.

367

Switch

- Veja o exemplo:

```
switch ($i) {
case 0:
    print "i é igual a zero";
case 1:
    print "i é igual a um";
case 2:
    print "i é igual a dois";
        }
```

- No exemplo acima, se $i for igual a zero, os três comandos "print" serão executados.
- Se $i for igual a 1, os dois últimos "print" serão executados.
- O comando só funcionará da maneira desejada se $i for igual a 2.

368

Switch

- Em outras linguagens que implementam o comando switch, ou similar, os valores a serem testados só podem ser do tipo inteiro.
- Em PHP é permitido usar valores do tipo string como elementos de teste do comando switch.
- O exemplo abaixo funciona perfeitamente:

```
switch ($s) {
case "casa":
    print "A casa é amarela";
case "arvore":
    print "a árvore é bonita";
case "lampada":
    print "joao apagou a lampada";
}
```

369

WHILE

- O while é o comando de repetição (laço) mais simples.
- Ele testa uma condição e executa um comando, ou um bloco de comandos, até que a condição testada seja falsa.
- Assim como o if, o while também possui duas sintaxes alternativas:

```
while (<expressao>)
  <comando>;

while (<expressao>):
  <comando>;
  . . .
  <comando>;
endwhile;
```

370

While

- A expressão só é testada a cada vez que o bloco de instruções termina, além do teste inicial.
- Se o valor da expressão passar a ser false no meio do bloco de instruções, a execução segue até o final do bloco.
- Se no teste inicial a condição for avaliada como false, o bloco de comandos não será executado.
- O exemplo a seguir mostra o uso do while para imprimir os números de 1 a 10:

```
$i = 1;
while ($i <=10)
  print $i++;
```

-

371

do ... while

- O laço do..while funciona de maneira bastante semelhante ao while, com a simples diferença que a expressão é testada ao final do bloco de comandos.
- O laço do..while possui apenas uma sintaxe, que é a seguinte:

```
do {
   <comando>
   . . .
   <comando>
} while (<expressao>);
```

- O exemplo utilizado para ilustrar o uso do while pode ser feito da seguinte maneira utilizando o do.. while:

```
$i = 0;
do {
  print ++$i;
} while ($i < 10);
```

372

for

- O tipo de laço mais complexo é o for. Para os que programam em C, C++ ou Java, a assimilação do funcionamento do for é natural.
- Mas para aqueles que estão acostumados a linguagens como Pascal, há uma grande mudança para o uso do for.
- As duas sintaxes permitidas são:

```
for (<inicializacao>;<condicao>;<incremento>)
   <comando>;

for (<inicializacao>;<condicao>;<incremento>) :
   <comando>;
   . . .
   <comando>;
endfor;
```

373

for

- As três expressões que ficam entre parênteses têm as seguintes finalidades:
- **Inicialização**: comando ou sequencia de comandos a serem realizados antes do inicio do laço. Serve para inicializar variáveis.
- **Condição**: Expressão booleana que define se os comandos que estão dentro do laço serão executados ou não. Enquanto a expressão for verdadeira (valor diferente de zero) os comandos serão executados.
- **Incremento**: Comando executado ao final de cada execução do laço.

374

for

- Um comando for funciona de maneira semelhante a um while escrito da seguinte forma:

```
<inicializacao>
while (<condicao>) {
comandos
...
<incremento>
}
```

- Equivale a:

```
for (<inicializacao>;<condicao>;<incremento>)
  <comando>;
```

375

Break

- O comando break pode ser utilizado em laços de do, for e while, além do uso já visto no comando switch.
- Ao encontrar um break dentro de um desses laços, o interpretador PHP para imediatamente a execução do laço, seguindo normalmente o fluxo do script.

```
while ($x > 0) {
  ...
  if ($x == 20) {
    echo "erro! x = 20";
    break;
  ...
}
```

- No trecho de código acima, o laço while tem uma condição para seu término normal ($x <= 0), mas foi utilizado o break para o caso de um término não previsto no início do laço. Assim o interpretador seguirá para o comando seguinte ao laço.

376

Continue

- O comando continue também deve ser utilizado no interior de laços, e funciona de maneira semelhante ao break, com a diferença que o fluxo ao invés de sair do laço volta para o início dele. Vejamos o exemplo:

```
for ($i = 0; $i < 100; $i++) {
  if ($i % 2) continue;
  echo " $i ";
}
```

- O exemplo acima é uma maneira ineficiente de imprimir os números pares entre 0 e 99. O que o laço faz é testar se o resto da divisão entre o número e 2 é 0.
- Se for diferente de zero (valor lógico true) o interpretador encontrará um continue, que faz com que os comandos seguintes do interior do laço sejam ignorados, seguindo para a próxima iteração.

377

Funções

- A sintaxe básica para definir uma função é:

```
function nome_da_função([arg1, arg2, arg3]) {
    Comandos;
    ... ;
    [return <valor de retorno>];
}
```

- Qualquer código PHP válido pode estar contido no interior de uma função. Como a checagem de tipos em PHP é dinâmica, o tipo de retorno não deve ser declarado, sendo necessário que o programador esteja atento para que a função retorne o tipo desejado.
- É recomendável que esteja tudo bem documentado para facilitar a leitura e compreensão do código.

378

Função

- Para efeito de documentação, utiliza-se o seguinte formato de declaração de função:

```
tipo function nome_da_funcao(tipo arg1, tipo arg2, ...);
```

- Este formato só deve ser utilizado na documentação do script, pois o PHP não aceita a declaração de tipos.
- Isso significa que em muitos casos o programador deve estar atento ao tipos dos valores passados como parâmetros, pois se não for passado o tipo esperado não é emitido nenhum alerta pelo interpretador PHP, já que este não testa os tipos.

379

Valor de Retorno

- Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum.
- Não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como listas ou arrays.

380

Argumentos

- É possível passar argumentos para uma função.
- Eles devem ser declarados logo após o nome da função, entre parênteses, e tornam-se variáveis pertencentes ao escopo local da função.
- A declaração do tipo de cada argumento também é utilizada apenas para efeito de documentação.

```
function imprime($texto){
   echo $texto;
}

imprime("teste de funções");
```

381

Passagem de Parâmetro por Referência

- Normalmente, a passagem de parâmetros em PHP é feita por valor, ou seja, se o conteúdo da variável for alterado, essa alteração não afeta a variável original.

```
function mais5($numero) {
   $numero += 5;
}

$a = 3;
mais5($a); //$a continua valendo 3
```

- No exemplo acima, como a passagem de parâmetros é por valor, a função mais5 é inútil, já que após a execução sair da função o valor anterior da variável é recuperado.
- Se a passagem de valor fosse feita por referência, a variável $a teria 8 como valor.

382

Passagem de Parâmetro por Referência

- O que ocorre normalmente é que ao ser chamada uma função, o interpretador salva todo o escopo atual, ou seja, os conteúdos das variáveis.
- Se uma dessas variáveis for passada como parâmetro, seu conteúdo fica preservado, pois a função irá trabalhar na verdade com uma cópia da variável.
- Porém, se a passagem de parâmetros for feita por referência, toda alteração que a função realizar no valor passado como parâmetro afetará a variável que o contém.
- Há duas maneiras de fazer com que uma função tenha parâmetros passados por referência: indicando isso na declaração da função, o que faz com que a pasagem de parâmetros sempre seja assim; e também na própria chamada da função. Nos dois casos utiliza-se o modificador "&".

383

Passagem de Parâmetro por Referência

```
function mais5(&$num1, $num2) {
  $num1 += 5;
  $num2 += 5;
}

$a = $b = 1;
mais5($a, $b); /* Neste caso, só $num1 terá seu valor alterado, pois a
   passagem por referência está definida na declaração da função. */


mais5($a, &$b); /* Aqui as duas variáveis terão seus valores alterados.
   */
```

384

Argumentos com valores pré-definidos (default)

- Em PHP é possível ter valores *default* para argumentos de funções, ou seja, valores que serão assumidos em caso de nada ser passado no lugar do argumento.
- Quando algum parâmetro é declarado desta maneira, a passagem do mesmo na chamada da função torna-se opcional.

```
function teste($cn = "testando") {
   echo $vivas;
}
teste(); // imprime "testando"
teste("outro teste"); // imprime "outro teste"
```

- É bom lembrar que quando a função tem mais de um parâmetro, o que tem valor *default* deve ser declarado por último:

385

Argumentos com valores pré-definidos (default)

```
function teste($figura = circulo, $cor) {
  echo "a figura é um ", $figura, " de cor " $cor;
}
teste(azul);
/* A função não vai funcionar da maneira esperada, ocorrendo um erro
   no interpretador. A declaração correta é: */

function teste2($cor, $figura = circulo) {
  echo "a figura é um ", $figura, " de cor " $cor;
}
teste2(azul);
/* Aqui a funcao funciona da maneira esperada, ou seja, imprime o
   texto: "a figura é um círculo de cor azul" */
```

386

Acessando Banco de Dados

- Para acessar bases de dados num servidor, é necessário antes estabelecer uma conexão.
- Para isso, deve ser utilizado o comando, **`odbc_connect`** ou o **`odbc_pconnect`**. A diferença entre os dois comandos é que o **`odbc_pconnect`** estabelece uma conexão permanente, ou seja, que não é encerrada ao final da execução do script.
- As assinaturas dos dois comandos são semelhantes, como pode ser verificado a seguir:
  - `int` **`odbc_connect`**`(string` *`dsn`*`, string` *`user`*`, string` *`password`*`, int` *`[cursor_type]`*`);`
  - `int` **`odbc_pconnect`**`(string` *`dsn`*`, string` *`user`*`, string` *`password`*`, int` *`[cursor_type]`*`);`
- O valor de retorno é um inteiro que identifica a conexão, ou falso se a conexão falhar.

387

odbc_connect

int **odbc_connect**(string *dsn*, string *user*, string *password*, int *[cursor_type]*);

- Retorna um identificador ODBC de conexão ou 0 em caso de erro.
- O identificador retornado por esta função é necessário o em outras funções ODBC.
- Pode-se ter muitas conexões abertas ao mesmo tempo.
- O quarto parâmetro é opcional. Não é necessário, mas pode ser útil para solucionar problemas com alguns drivers.

388

odbc_connect

- Alguns Drivers ODBC que executam um stored procedure complexo, pode falhar com uma mensagem como: "Cannot open a cursor on a stored procedure that has anything other than a single select statement in it"
- Utilizando SQL_CUR_USE_ODBC pode-se evitar este erro. Também alguns drivers não suportam o parâmetro *row_number* no comando **odbc_fetch_row.** SQL_CUR_USE_ODBC pode ajudar neste caso.
- As constantes abaixo são definidas para o parametro *cursortype.*

  SQL_CUR_USE_IF_NEEDED
  SQL_CUR_USE_ODBC
  SQL_CUR_USE_DRIVER
  SQL_CUR_DEFAULT

389

Acessando Banco de Dados

- Uma conexão estabelecida com o comando odbc_connect é encerrada ao final da execução do script. Para encerrá-la antes disso deve ser utilizado o comando odbcl_close, que tem a seguinte assinatura:

  void **odbc_close**(int *connection_id*);
  void **odbc_close_all**(void);

- IMPORTANTE: o comando odbc_close não encerra conexões estabelecidas com o comando odbc_pconnect.
- Nota: Estas funções falharão se houver transações abertas em uma conexão. Esta conexão permanecerá aberta neste caso.

390

Conversando com a base de dados

Depois de estabelecida a conexão, é preciso executar algum comando SQL.

int **odbc_execute**(int *result_id*, array *[parameters_array]*);

- Executa uma declaração preparada com odbc_prepare. Retorna Verdadeiro se a execução for bem sucedido ou falso en caso contrário. O parameters_array, só precisa ser dado se você realmente tiver parâmetros em sua declaração.

int **odbc_prepare**(int *connection_id*, string *query_string*);

- Prepara um comando para execução. Retorna falso em caso de erro senão retorna um identificador ODBC se o comando SQL for preparado com sucesso. O idenfiticador pode ser utilizado no comando odbc_execute.

391

Exemplo

```
<?
   $conn = odbc_connect("hiromi", "", "");
   $stmt ="UPDATE nome SET nome = 'CAIO'  WHERE
CODIGO = 2";
   $prepare=odbc_prepare($conn,$stmt);
   $resp = odbc_execute($prepare);
   if ($resp)
        echo (“Atualizado com sucesso”);
   else
        echo (“Falha na operação”);
   odbc_close($com);
?>
```

392

int **odbc_exec**(int *connection_id*, string *query_string*);

- Prepara e executa um comando SQL.
- Retorna FALSO quando ocorre um erro ou retorna um identificador ODBC se o comando SQL é executado com sucesso.
- ***odbc_exec*** manda um comando SQL para o Servidor de Banco de Dados, especificado por ***connection_id***. Este parametro deve ser um identificador válido que é determinado por ***odbc_connect***.

```
<?
   $conn = odbc_connect("hiromi", "", "");
   $stmt ="UPDATE nome SET nome = 'CAIO'  WHERE
CODIGO = 2";
   $resp=odbc_exec($conn,$stmt);
?>
```

393

Recuperando Informações

- Após executar uma consulta SQL no Banco de Dados é necessário transportar os dados da consulta para variáveis PHP.
- Percorre-se o cursor com o comando FETCH.

int **odbc_fetch_into**(int *result_id*, int *[rownumber]*, array *result_array*);

- O retorna o número de colunas do resultado ou false em caso de erro. r*esult_array* deve ser passado por referencia, mas pode ser qualquer variável, uma vez que pode ser convertido dinamicamente. O vetor conterá valores que começam com índice 0.

394

int **odbc_fetch_row**(int *result_id*, int *[row_number]*);

- Se ***odbc_fetch_row*** estiver correto (existir uma linha), retorna verdadeiro, senão existir linha retorna falso.
- ***odbc_fetch_row*** traz um linha de dados que retorna do comando ***odbc_exec.*** Depois de chamar a função ***odbc_fetch_row*** é necessário chamar a função ***odbc_result*** para que os campos possam ser acessados.
- Se o parâmetro *row_number* não for especificado, ***odbc_fetch_row*** tentará buscar o próxima linhas do cursor resultado. Chamada para a função ***odbc_fetch_row*** com ou sem o parâmetro *row_number* pode ser alternado.
- Para percorrer através do resultado da consulta várias vezes, pode-se chamar a função ***odbc_fetch_row*** com *row_number* 1 e então continuar com ***odbc_fetch_row*** sem o parâmetro.
- Se o driver do Banco de Dados não suportar busca por linha, *odbc_row* é ignorado.

395

string **odbc_result**(int *result_id*, mixed *field*);

- Retorna o conteúdo do campo.
- *field* pode ser um valor inteiro contendo o número da coluna que você deseja, ou pode ser um texto contendo o nome do campo.

```
$item_3 = odbc_result($Query_ID, 3 );
$item_val = odbc_result($Query_ID, "val");
```

- A primeira chamada de ***odbc_result*** retorna o valor do terceiro campo do cursor que resultou em $Query. A segunda chamada de função ***odbc_result*** retorna o valor do campo chamado “val” no cursor corrente.
- Um erro ocorre se o número da coluna for menor que um ou excede o número de campos do registro. Assim como chamar um campo com nome inválido para o registro consultado.

396

```
<?
   $con = odbc_connect("CAFESODBC");
   $rescid = odbc_exec($con,"SELECT * FROM UNIDADE where
     UNIDADEID =".$p_unidade);
   $fetcid = odbc_fetch_row($rescid );
   $cidade  = odbc_result($rescid, 2);
   $ud_cep  = odbc_result($rescid, 4);
   $ud_end  = odbc_result($rescid, 3);
   echo '<INPUT TYPE="hidden" NAME="p_unidade"
     VALUE="'.$p_unidade.'">';
   echo '<INPUT TYPE="hidden" NAME="p_cid" VALUE="'.$cidade.'">';
   echo '<INPUT TYPE="hidden" NAME="p_datacad"
   VALUE="'.$datacad.'">';
   odbc_close($con );
?>
```

397

- int **odbc_result_all**(int *result_id*, string *[format]*);
- Retorna o número de linhas do resultado ou falso se ocorrer um erro.
- ***odbc_result_all*** imprimirá todas as linhas do identificador produzido por ***odbc_exec.***
- O resultado será impresso em uma tabela HTML. O argumento *format* pode ser definido para definir parâmetros adicionais da tabela HTML.

```
<?
   $con = odbc_connect("CAFESODBC");
   $rescid = odbc_exec($con,"SELECT * FROM UNIDADE where
     UNIDADEID =".$p_unidade);
   $fetcid = odbc_fetch_row($rescid );
   $cidade  = odbc_result_all($rescid);
   odbc_close($con );
?>
```

398

string **odbc_fieldname**(int *result_id*, int *field_number*);

- ***odbc_field_name*** retorna o nome do campo ocupado por pela coluna definida em *field_number* no cursor da consulta ODBC.
- Os números dos campos começam com 1. Retorna falso em caso de erro.

string **odbc_field_type**(int *result_id*, int *field_number*);

- ***odbc_field_type*** retornará um tipo de campo SQL referenciado pelo número dado pelo identificador ODBC.

int **odbc_field_len**(int *result_id*, int *field_number*);

- ***odbc_field_len*** retornará o tamanho do campo referenciado pelo número dado pelo identificador ODBC.

399

Classe

- Uma classe é um conjunto de variáveis e funções relacionadas a essas variáveis.
- Uma vantagem da utilização de programação orientada a objetos é poder usufruir do recurso de encapsulamento de informação.
- Com o encapsulamento o usuário de uma classe não precisa saber como ela é implementada, bastando para a utilização conhecer a interface, ou seja, as funções disponíveis.
- Uma classe é um tipo, e portanto não pode ser atribuída a uma variável.

400

Classe

- Para definir uma classe, deve-se utilizar a seguinte sintaxe:

```
class Nome_da_classe {
   var $variavel1;
   var $variavel2;
   function funcao1 ($parametro) {
    /* === corpo da função === */
   }
}
```

401

Objeto

- Como foi dito anteriormente, classes são tipos, e não podem ser atribuídas a variáveis.
- Variáveis do tipo de uma classe são chamadas de objetos, e devem ser criadas utilizando o operador new, seguindo o exemplo abaixo:

```
$variavel = new $nome_da_classe;
```

- Para utilizar as funções definidas na classe, deve ser utilizado o operador "->", como no exemplo:

```
$variavel->funcao1(parametro);
```

402

A variável $this

- Na definição de uma classe, pode-se utilizar a variável $this, que é o próprio objeto.
- Assim, quando uma classe é instanciada em um objeto, e uma função desse objeto na definição da classe utiliza a variável $this, essa variável significa o objeto que estamos utilizando.
- Como exemplo da utilização de classes e objetos, podemos utilizar a classe conta, que define uma conta bancária bastante simples, com funções para ver saldo e fazer um crédito.

403

Exemplo

```
class conta {
   var $saldo;
   function saldo() {
     return $this->saldo;
   }
   function credito($valor) {
     $this->saldo += $valor;
   }
}

$minhaconta = new conta;
          $minhaconta->saldo();   // a variavel interna não foi
          // inicializada, e não contém
          // valor algum
$minhaconta->credito(50);
$minhaconta->saldo(); // retorna 50
```

404

Sub classes

- Uma classe pode ser uma extensão de outra. Isso significa que ela herdará todas as variáveis e funções da outra classe, e ainda terá as que forem adicionadas pelo programador. Em PHP não é permitido utilizar herança múltipla, ou seja, uma classe pode ser extensão de apenas uma outra.Para criar uma classe extendida, ou derivada de outra, deve ser utilizada a palavra reservada extends, como pode ser visto no exemplo seguinte:

```
class novaconta extends conta {
  var $num;
  function numero() {
    return $this->num;
  }
}
```

- A classe acima é derivada da classe conta, tendo as mesmas funções e variáveis, com a adição da variável $numero e a função numero().

405

Construtores

- Um construtor é uma função definida na classe que é automaticamente chamada no momento em que a classe é instanciada (através do operador new).
- O construtor deve ter o mesmo nome que a classe a que pertence. Veja o exemplo:

```
class conta {
  var $saldo;
  function conta () {
    $this.saldo = 0;
  }
  function saldo() {
    return $this->saldo;
  }
  function credito($valor) {
    $this->saldo += $valor;
  }
}
```

406

Construtores

- Podemos perceber que a classe conta agora possui um construtor, que inicializa a variável $saldo com o valor 0.
- Um construtor pode conter argumentos, que são opcionais, o que torna esta ferramenta mais poderosa. No exemplo acima, o construtor da classe conta pode receber como argumento um valor, que seria o valor inicial da conta.
- Vale observar que para classes derivadas, o construtor da classe pai não é automaticamente herdado quando o construtor da classe derivada é chamado.

407

Cookies

- Cookies são variáveis gravadas no cliente(browser) por um determinado site.
- Somente o site que gravou o cookie pode ler a informação contida nele.
- Este recurso é muito útil para que determinadas informações sejam fornecidas pelo usuário apenas uma vez.
- Exemplos de utilização de cookies são sites que informam a quantidade de vezes que você já visitou, ou alguma informação fornecida numa visita anterior.
- Existem cookies persistentes e cookies de sessão.
- Os persistentes são aqueles gravados em arquivo, e que permanecem após o browser ser fechado, e possuem data e hora de expiração.

408

Cookies

- Os cookies de sessão não são armazenados em disco e permanecem ativos apenas enquanto a sessão do browser não for encerrada.
- Por definição, existem algumas limitações para o uso de cookies, listadas a seguir:
  - 300 cookies no total
  - 4 kilobytes por cookie.
  - 20 cookies por servidor ou domínio.

409

Gravando cookies

- Para gravar cookies no cliente, deve ser utilizada a função setcookie, que possui a seguinte assinatura:

int setcookie(string nome, string valor, int exp, string path, string dominio, int secure);

onde:

nome: nome do cookie;

valor: valor armazenado no cookie;

exp: data de expiração do cookie (opcional), no formato Unix. Se não for definida, o cookie será de sessão;

path: path do script que gravou o cookie;

dominio: domínio responsável pelo cookie;

secure: se tiver valor 1, indica que o cookie só pode ser transmitido por uma conexão segura (https).

410

Gravando cookies

- Observações:
- Um cookie não pode ser recuperado na mesma página que o gravou, a menos que esta seja recarregada pelo browser.
- Múltiplas chamadas à função setcookie serão executadas em ordem inversa;
- Cookies só podem ser gravados antes do envio de qualquer informação para o cliente. Portanto todas as chamadas à função setcookie devem ser feitas antes do envio de qualquer header ou texto.

411

Lendo cookies gravados

- Os cookies lidos por um script PHP ficam armazenados em duas variáveis. no array $HTTP_COOKIE_VARS[], tendo como índice a string do nome do cookie, e numa variável cujo nome é o mesmo do cookie, precedido pelo símbolo $.
- Um cookie que foi gravado numa página anterior pelo seguinte comando:
  setcookie("teste", "meu cookie");
- Pode ser lida pela variável
  $HTTP_COOKIE_VARS["teste"]
- ou pela variável
  $teste

412

Exemplo COOKIES

```
<?
        setcookie("teste", 1);
        $aux = getdate();
        $datahoje = $aux["mday"]."/".$aux["mon"]."/".$aux["year"];
        $datahoje .= " - ".$aux["hours"].":".$aux["minutes"].":".$aux["seconds"];
        $UltimoAcesso = $acesso;
        setcookie("acesso",$datahoje);
        echo $UltimoAcesso;
?>
<html>
    <a href=bolacha.php>
    Página Bolacha                    </a>
</html>
```

413

Funções relacionadas a HTML

string htmlspecialchars(string str);

- Retorna a string fornecida, substituindo os seguintes caracteres:
  - & para '&'
  - " para '"'
  - < para '<'
  - > para '>'

string htmlentities(string str);

- Funciona de maneira semelhante ao comando anterior, mas de maneira mais completa, pois converte todos os caracteres da string que possuem uma representação especial em html, como por exemplo:
  - º para 'º'
  - ª para 'ª'
  - á para 'á'
  - ç para 'ç'

414

string nl2br(string str);

- Retorna a string fornecida substituindo todas as quebras de linha (“\n”) por quebras de linhas em html (“
”).

```
echo nl2br(“Caio\nNakashima\n”);
Imprime:
Caio<br>Nakashima<br>
```

string strip_tags(string str);

- Retorna a string fornecida, retirando todas as tags html e/ou PHP encontradas.

```
strip_tags('<a href="teste1.php3">testando</a><br>');
```

Retorna a string “testando”

415

array get_meta_tags(string arquivo);

- Abre um arquivo html e percorre o cabeçalho em busca de “meta” tags, retornando num array todos os valores encontrados.
- No arquivo teste.html temos:

```
...
<head>
<meta name="author" content="jose">
<meta name="tags" content="php3 documentation">
...
</head><!-- busca encerra aqui -->
...
```

- a execução da função:

```
get_meta_tags(“teste.html”);
```

- retorna o array:

```
array(“author”=>”jose”,”tags”=>"php3 documentation");
```

416

string urlencode(string str);

- Retorna a string fornecida, convertida para o formato urlencode. Esta função é útil para passar variáveis para uma próxima página.

string urldecode(string str);

- Funciona de maneira inversa a urlencode, desta vez decodificando a string fornecida do formato urlencode para texto normal.

417

Funções relacionadas a arrays

string implode(string separador, array partes);

string join(string separador, array partes);

- As duas funções são idênticas. Retornam uma string contendo todos os elementos do array fornecido separados pela string também fornecida.

```
$partes = array("a", "casa número", 13, "é azul");
$inteiro = join(" ",$partes);
```

$inteiro passa a conter a string:

"a casa número 13 é azul"

418

array split(string padrao, string str, int [limite]);

- Retorna um array contendo partes da string fornecida separadas pelo padrão fornecido, podendo limitar o número de elementos do array.

  $data = "11/14/1975";

  $data_array = split("/",$data);

  – O código acima faz com que a variável $data_array receba o valor:

  – array(11,14,1975);

array explode(string padrao, string str);

- Funciona de maneira bastante semelhante à função split, com a diferença que não é possível estabelecer um limite para o número de elementos do array.

419

Comparações entre strings

int similar_text(string str1, string str2, double [porcentagem]);

- Compara as duas strings fornecidas e retorna o número de caracteres coincidentes. Opcionalmente pode ser fornecida uma variável, passada por referência *(ver tópico sobre funções)*, que receberá o valor percentual de igualdade entre as strings. Esta função é *case sensitive*, ou seja, maiúsculas e minúsculas são tratadas como diferentes.

  $num = similar_text("teste", "testando",&$porc);

  As variáveis passam a ter os seguintes valores:

  $num == 4; $porc == 61.538461538462

int strcmp(string str1, string str2);

- Compara as duas strings e retorna 0 (zero) se forem iguais, um valor maior que zero se str1 > str2, e um valor menor que zero se str1 < str2. Esta função é *case insensitive*, ou seja, maiúsculas e minúsculas são tratadas como iguais.

420

int strcasecmp(string str1, string str2);

- Funciona de maneira semelhante à função strcasecmp, com a diferença que esta é *case sensitive*, ou seja, maiúsculas e minúsculas são tratadas como diferentes.

string strstr(string str1, string str2);

string strchr(string str1, string str2);

- As duas funções são idênticas. Procura a primeira ocorrência de str2 em str1. Se não encontrar, retorna uma string vazia, e se encontrar retorna todos os caracteres de str1 a partir desse ponto.

  strstr(”Caio Nakashima", ”Naka"); // retorna “Nakashima”

string strstr(string str1, string str2);

- Funciona de maneira semelhante à função strstr, com a diferença que esta é *case insensitive*, ou seja, maiúsculas e minúsculas são tratadas como iguais.

421

int strpos(string str1, string str2, int [offset]);

- Retorna a posição da primeira ocorrência de str2 em str1, ou zero se não houver. O parâmetro opcional offset determina a partir de qual caracter de str1 será efetuada a busca. Mesmo utilizando o offset, o valor de retorno é referente ao início de str1.

int strrpos(string haystack, char needle);

- Retorna a posição da última ocorrência de str2 em str1, ou zero se não houver.

422

Funções para edição de strings

string chop(string str);

- Retira espaços e linhas em branco do final da string fornecida.

  chop(" Teste \n \n "); // retorna " Teste"

string ltrim(string str);

- Retira espaços e linhas em branco no início da string fornecida.

  ltrim(" Teste \n \n "); // retorna "Teste \n \n"

string trim(string str);

- Retira espaços e linhas em branco do início e do final da string fornecida.

  trim(" Teste \n \n "); // retorna "Teste"

423

string strrev(string str);

- Retorna a string fornecida invertida.

  strrev("Teste"); // retorna "etseT"

string strtolower(string str);

- Retorna a string fornecida com todas as letras minúsculas.

  strtolower("Teste"); // retorna "teste"

string strtoupper(string str);

- Retorna a string fornecida com todas as letras maiúsculas.

  strtolower("Teste"); // retorna "TESTE"

424

string ucfirst(string str);

- Retorna a string fornecida com o primeiro caracter convertido para letra maiúscula.

  ucfirst("teste de funcao"); // retorna "Teste de funcao"

string ucwords(string str);

- Retorna a string fornecida com todas as palavras iniciadas por letras maiúsculas.

  ucwords("teste de funcao"); // retorna "Teste De Funcao"

string str_replace(string str1, string str2, string str3);

- Altera todas as ocorrências de str1 em str3 pela string str2.

425

Funções diversas

string chr(int ascii);

- Retorna o caracter correspondente ao código ASCII fornecido.

int ord(string string);

- Retorna o código ASCII correspontente ao caracter fornecido.

echo(string arg1, string [argn]...);

- Imprime os argumentos fornecidos.

print(string arg);

- Imprime o argumento fornecido.

int strlen(string str);

- Retorna o tamanho da string fornecida.

426

array array(...);

- É a função que cria um array a partir dos parâmetros forncidos. É possível fornecer o índice de cada elemento. Esse índice pode ser um valor de qualquer tipo, e não apenas de inteiro. Se o índice não for fornecido o PHP atribui um valor inteiro sequencial, a partir do 0 ou do último índice inteiro explicitado. Vejamos alguns exemplos:

  $teste = array("um","dois","tr"=>"tres",5=>"quatro","cinco");

  Temos o seguinte mapeamento:

  0 => "um" (0 é o primeiro índice, se não houver um explicito)

  1 => "dois" (o inteiro seguinte)

  tr => "tres"

  5 => "quatro" (valor explicitado)

  6 => "cinco" (o inteiro seguinte ao último atribuído, e não o próximo valor, que seria 2)

427

- Exemplo 2

  $teste = array("um",6=>"dois","tr"=>"tres",5=>"quatro","cinco");

  Temos o seguinte mapeamento:

  0 => "um"

  6 => "dois"

  tr => tres

  5 => "quatro" (seria 7, se não fosse explicitado)

  7 => "cinco" (seria 6, se não estivesse ocupado)

- Em geral, não é recomendável utilizar arrays com vários tipos de índices, já que isso pode confundir o programador. No caso de realmente haver a necessidade de utilizar esse recurso, deve-se ter bastante atenção ao manipular os índices do array.

428

array range(int minimo, int maximo);

- A função range cria um array cujos elementos são os inteiros pertencentes ao intervalo fornecido, inclusive. Se o valor do primeiro parâmetro for maior do que o do segundo, a função retorna false (valor vazio).

void shuffle(array &arr);

- Esta função "embaralha" o array, ou seja, troca as posições dos elementos aleatoriamente e não retorna valor algum.

int sizeof(array arr);

- Retorna um valor inteiro contendo o número de elementos de um array. Se for utilizada com uma variável cujo valor não é do tipo array, retorna 1. Se a variável não estiver setada ou for um array vazio, retorna 0.

429

Funções de "navegação"

- Toda variável do tipo array possui um ponteiro interno indicando o próximo elemento a ser acessado no caso de não ser especificado um índice.
- As funções seguintes servem para modificar esse ponteiro, permitindo assim percorrer um array para verificar seu conteúdo (chaves e elementos).

mixed reset(array arr);

- Seta o ponteiro interno para o primeiro elemento do array, e retorna o conteúdo desse elemento.

mixed end(array arr);

- Seta o ponteiro interno para o último elemento do array, e retorna o conteúdo desse elemento.

430

mixed next(array arr);

- Seta o ponteiro interno para o próximo elemento do array, e retorna o conteúdo desse elemento.
- Obs.: esta não é uma boa função para determinar se um elemento é o último do array, pois pode retornar false tanto no final do array como no caso de haver um elemento vazio.

mixed prev(array arr);

- Seta o ponteiro interno para o elemento anterior do array, e retorna o conteúdo desse elemento. Funciona de maneira inversa a next.

mixed pos(array arr);

- Retorna o conteúdo do elemento atual do array, indicado pelo ponteiro interno.

431

mixed key(array arr);

- Funciona de maneira bastante semelhante a pos, mas ao invés de retornar o elemento atual indicado pelo ponteiro interno do array, retorna seu índice.

array each(array arr);

- Retorna um array contendo o índice e o elemento atual indicao pelo ponteiro interno do array. o valor de retorno é um array de quatro elementos, cujos índices são 0, 1, "key" e "value". Os elementos de índices 0 e "key" armazenam o índice do valor atual, e os elementos de índices 1 e "value" contém o valor do elemento atual indicado pelo ponteiro.
- Esta função pode ser utilizada para percorrer todos os elementos de um array e determinar se já foi encontrado o último elemento, pois no caso de haver um elemento vazio, a função não retornará o valor false. A função each só retorna false depois q o último elemento do array foi encontrado.

432

Exemplo:

```
/*função que percorre todos os elementos de um array e imprime seus
   índices e valores */
function imprime_array($arr) {
  reset($arr);
  while (list($chave,$valor) = each($arr))
    echo "Chave: $chave. Valor: $valor";
}
```

433

Funções de ordenação

- São funções que servem para arrumar os elementos de um array de acordo com determinados critérios.
- Estes critérios são: manutenção ou não da associação entre índices e elementos; ordenação por elementos ou por índices; função de comparação entre dois elementos.

void sort(array &arr);

- A função mais simples de ordenação de arrays. Ordena os elementos de um array em ordem crescente, sem manter os relacionamentos com os índices.

void rsort(array &arr);

- Funciona de maneir ainversa à função sort. Ordena os elementos de um array em ordem decrescente, sem manter os relacionamentos com os índices.

434

void asort(array &arr);

- Tem o funcionamento bastante semelhante à função sort. Ordena os elementos de um array em ordem crescente, porém mantém os relacionamentos com os índices.

void arsort(array &arr);

- Funciona de maneira inversa à função asort. Ordena os elementos de um array em ordem decrescente e mantém os relacionamentos dos elementos com os índices.

void ksort(array &arr);

- Função de ordenação baseada nos índices. Ordena os elementos de um array de acordo com seus índices, em ordem crescente, mantendo os relacionamentos.

435

void usort(array &arr, function compara);

- Esta é uma função que utiliza outra função como parâmetro. Ordena os elementos de um array sem manter os relacionamentos com os índices, e utiliza para efeito de comparação uma função definida pelo usuário, que deve comparar dois elementos do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

void uasort(array &arr, function compara);

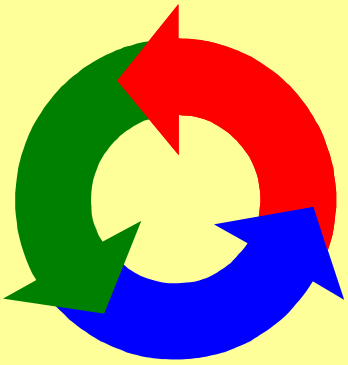
- Esta função também utiliza outra função como parâmetro. Ordena os elementos de um array e mantém os relacionamentos com os índices, utilizando para efeito de comparação uma função definida pelo usuário, que deve comparar dois elementos do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

436

void uksort(array &arr, function compara);

- Esta função ordena o array através dos índices, mantendo os relacionamentos com os elementos., e utiliza para efeito de comparação uma função definida pelo usuário, que deve comparar dois índices do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

437



438

Links e material para consulta

- Apache - http://www.apache.org/
- Apache JServ - http://java.apache.org/
- OpenSSL - http://www.openssl.org/
- PHP - http://www.php.org/
- Mod_SSL - http://www.modssl.org/
- RSA - http://www.rsa.com/
- Unicamp - ftp://ftp.unicamp.br/
- Sun Java soft - http://www.javasoft.com/
- JavaScript - http://home.netscape.com/eng/mozilla/Gold/handbook/javascript/index.html
- Caio Nakashima - http://www.dainf.cefetpr.br/~nakashi

439