Apostila de Banco de Dados

Revisão de Abordagem Relacional

O modelo relacional representa o BD como uma coleção de relações. Cada relação pode ser vista como uma tabela e cada linha representa uma entidade ou um relacionamento.

- **Tabela:** é uma relação composta por um conjunto não ordenado de linhas/tuplas.
- **Colunas:** são os atributos das relações.
- **Domínio:** tipo de dados que pode aparecer em cada coluna.
- **Chave primária (PK):** é uma coluna ou uma combinação de colunas cujos valores distinguem uma linha das demais em uma tabela.
- **Chave estrangeira (FK):** é uma coluna cujos valores aparecem necessariamente na chave primária de outra tabela.

Um esquema de relação é representado por: **R (A1, A2, ..., An)**. Onde **R** é o nome da relação e **A1, A2, ..., An** é a lista dos atributos.

- n é o grau da relação.
- Dom(Ai) é o domínio do atributo Ai.

Por exemplo, o esquema de relação para a tabela abaixo é: Aluno (Nome, Matricula, Endereco, Idade, FoneRes, RG).

Nome	Matricula	Endereço	Idade	FoneRes	RG
João	03213-05	Rua x, 20	19	3424-0111	12345
José	00571-01	Av. B, 987	24	3421-9899	98765
Antônio	00976-03	Rua A, 032	22	3423-0987	16578

A manutenção da integridade de dados é um dos objetivos de um SGBD. Dados íntegros refletem corretamente a realidade. Uma restrição de integridade é uma regra de consistência de dados garantida pelo SGBD.

- **Restrição de domínio:** valor de um campo deve obedecer a definição de valores admitidos da coluna (domínio).
- **Restrição de vazio:** especifica quais colunas de uma tabela podem receber ou não valores nulos.
- **Restrição de chave:** define que os valores da chave primária e alternativa devem ser únicos e não vazios.
- **Restrição de integridade referencial:** define que os valores dos campos que aparecem em uma chave estrangeira devem aparecer na chave primária da tabela referenciada; essa restrição é utilizada para manter a consistência entre tuplas de duas relações.

Um esquema **S** de BD relacional contém várias relações **{R1, ..., Rm}** e um conjunto de restrições de integridade.

Mapeamento ER - Relacional

Um modelo ER pode ser implementado em diversos modelos relacionais. Os objetivos centrais da transformação de modelos são: bom desempenho (performance) e simplificação do desenvolvimento e da manutenção. Princípios a serem seguidos:

- Evitar junções (ter os dados necessários para uma consulta em uma única linha).
- Diminuir o número de chaves.
- Evitar campos opcionais (evitar desperdício de espaço ao gerar campos vazios).

❤ Regra 1 - Entidade Regular

Cada entidade é traduzida em uma tabela.

- Para cada entidade forte **E**, cria-se uma relação **R** e inclui todos atributos simples de E.

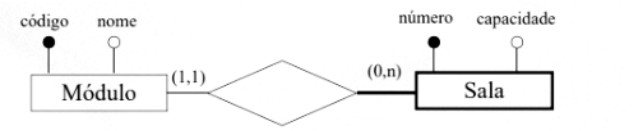
- Incluir os atributos simples que fizerem parte dos atributos compostos de E.
- Um dos atributos chaves de E é escolhido (grifado) como PK de R.

❤ Regra 2 - Entidade Fraca

Cada entidade é traduzida em uma tabela, com alguns detalhes:

- Criar uma relação R para a entidade fraca W.
- Incluir como chave estrangeira de W as chaves primárias da entidade proprietária de W.
- A chave primária de R é a combinação das chave(s) primária(s) da(s) entidade(s) proprietária(s) de W e a chave parcial de W.

Exemplo com as regras 1 e 2:



•*Modulo* (codigo:int, nome:string)

•***Sala*** **(codModulo:int, numero:int, capacidade:int)**

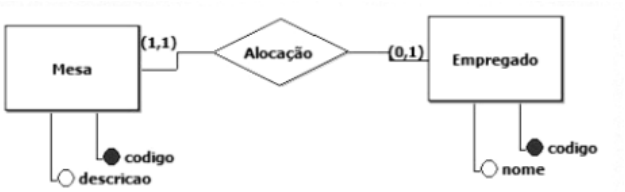
codModulo referencia *Modulo*

OBS: Informar a qual tabela a chave estrangeira referencia.

❤ Regra 3 - Relacionamento 1:1

Para mapear um relacionamento R binário 1:1, primeiro identifique as relações S e T que participam do relacionamento. Após isso, há três alternativas:

1. Escolher uma das relações do, e incluir como chave estrangeira de uma a chave primária da outra.
2. Criar uma relação para R.
3. Criar apenas uma relação com todos os atributos de S e T.



Dado o modelo acima, o mapeamento de acordo a cada uma das opções seria:

1° opção: a chave primária de uma das relações (Mesa) torna-se chave estrangeira da outra (Empregado).

Mesa (codigo:int, descricao:string)
Empregado(codigo:int, nomeEmpregado:string, cdMesa:int)
cdMesa referencia Mesa

2° opção: cria-se a relação Alocação

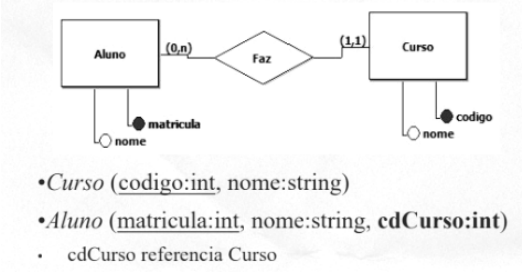
Mesa (codigo:int, descricao:string)
Empregado(codigo:int, nomeEmpregado:string)
Alocacao (cdMesa:int, cdEmp:int)
cdMesa referencia Mesa
cdEmp referencia Empregado

3° opção: cria-se uma relação para tudo

AlocacaoMesaEmp (cdMesa:int, descMesa:string, cdEmp:int, nomeEmp:string)

❤ Regra 4 - Relacionamento 1:n

- Identifique a relação que representa a entidade S que participa 1 vez no relacionamento.
- Inclua como chave estrangeira de S a chave primária da relação T que representa a outra entidade do relacionamento. (colocar do lado que tem o n)
- Inclua qualquer atributo simples do relacionamento 1:N em S.



❤ Regra 5 - Relacionamento n:n

- Para um relacionamento binário R n:n, cria-se uma nova relação S para representar R.
- Inclua como chave estrangeira em S, as chaves primárias das relações que participam do relacionamento. A combinação destas chaves formará a chave primária (PK) da relação S.
- Inclua qualquer atributo do relacionamento R em S.

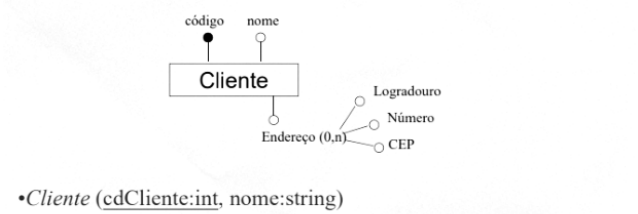
Esquema relacional correspondente:

```
Engenheiro (CodEng,Nome)
Projeto(CodProj,Título)
Atuação (CodEng,CodProj,Função)
  CodEng referencia Engenheiro
  CodProj referencia Projeto
```

OBS: É possível mapear o relacionamento 1:1 ou 1:n de maneira similar ao n:n.

❤ Regra 6 - Atributo Multivalorado

- Para cada atributo multivalorado A, cria-se uma nova relação R. R irá incluir o atributo A, além da chave primária K da relação que representa a entidade (ou relacionamento) que tem A como atributo.
- A chave primária de R é a combinação de A e K.
- Se o atributo multivalorado é composto, incluir seus componentes atômicos.



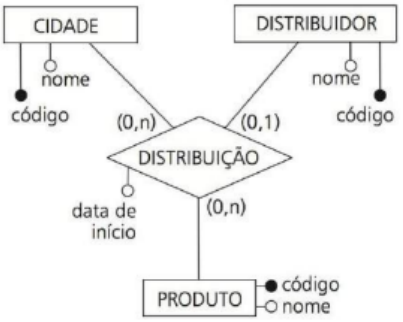
•*Cliente* (cdCliente:int, nome:string)

•*Endereço* (cdCliente:int, logradouro:string, número:int, CEP:string)

cdCliente referencia Cliente

❤ Regra 7 - Relacionamento n-ário

- Para o relacionamento n-ário R, cria-se uma nova relação S para representar R.
- Inclua como chave estrangeira em S, as chaves primárias das relações que representam as entidades que participam de R. Inclua também os atributos de R
- A chave primária de S é a combinação das chaves estrangeiras.



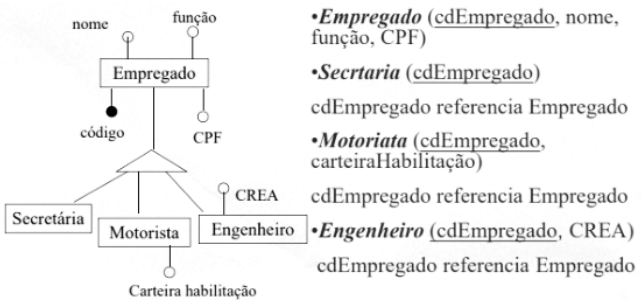

```
Produto(CodProd,Nome) [Cidade (CodCid,Nome)
Distribuidor(CodDistr,Nome)
Distribuição (CodProd,CodDistr,CodCid,DataDeInicio)
    CodProd referencia Produto
    CodDistr referencia Distribuidor
    CodCid referencia Cidade
```

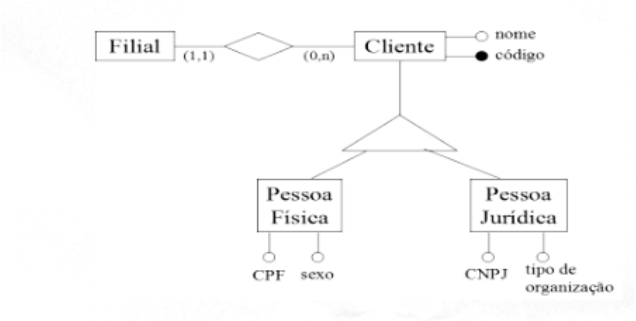
❤ Regra 8 - Generalização/Especialização

Para converter uma especialização com m subclasses e uma superclasse C, pode se usar as seguintes opções:

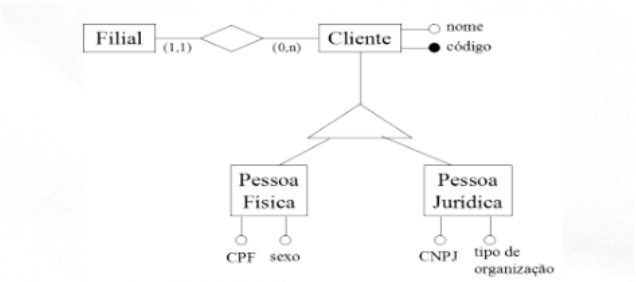
- **8A:** Criar uma relação para a superclasse C com todos seus atributos e sua PK; criar uma relação para cada uma das especializações com seus respectivos atributos, entre eles uma FK que é a PK da superclasse.
- **8B:** Criar apenas relações para as subclasses, contendo os seus atributos específicos e todos os atributos que existem na superclasse.
- **8C:** Criar uma única relação L com os atributos da superclasse C e de todas as subclasses.

Exemplos de mapeamento de herança 8A, 8B e 8C, respectivamente:





•*PessoaFisica* (codigo, nome, CPF, sexo)
•*PessoaJuridica* (codigo, nome, CNPJ, tipoOrg)



•*Cliente* (codigo, nome, CPF, sexo, CNPJ, tipoOrg)

Álgebra Relacional

O conjunto básico de operações para o modelo relacional é a álgebra relacional, que por sua vez, é uma linguagem de consultas procedural.

Assim, a álgebra relacional consiste em um conjunto de operações que tem como entrada uma ou duas relações e produz, como resultado, uma nova **relação**.

Importância da álgebra relacional:

- Alicerce formal para as operações do modelo relacional;
- Base para a implementação e otimização de consultas;
- Alguns de seus conceitos são incorporados em SGBDs.

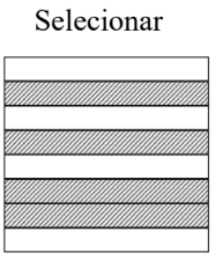
Principais operações: seleção, projeção, união, interseção, diferença, produto cartesiano e renomeação.

Classificação dos operadores:

- **Unários:** seleção e projeção
- **Binários:** união, diferença e produto cartesiano.

❤ Seleção (σ)

A operação **seleção** é usada para escolher um subconjunto das tuplas de uma relação que satisfaça uma condição de seleção. Pode-se considerar a seleção como um filtro.



Essa operação é representada pelo símbolo σ. A condição aparece subscrita a σ e o argumento da relação é dado entre parênteses.

$$\sigma_{\text{<condição seleção>}}(R)$$

Nos exemplos abaixo, estamos selecionando funcionários cujo departamento é 4 e funcionários que têm um salário maior do que R$30.000,00.

$$\sigma_{Dnr=4}(\text{FUNCIONARIO})$$
$$\sigma_{Salario>30.000}(\text{FUNCIONARIO})$$

A expressão booleana da condição contém operadores de comparação pertencentes ao conjunto {=, <, ≤, >, ≥, ≠} e é composta por uma série de cláusulas que podem ser conectadas com operadores como **AND**, **OR** e **NOT**.

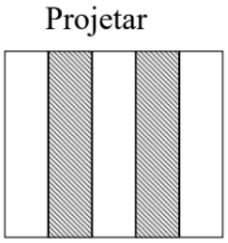
$$\sigma_{(\text{Dnr=4 AND Salario>25.000}) \text{ OR } (\text{Dnr=5 AND Salario>30.000})}(\text{FUNCIONARIO})$$

A operação de seleção é aplicada a cada tupla **individualmente**. Logo, as condições de seleção não podem envolver mais de uma tupla. Além disso, a seleção é comutativa, ou seja:

$$\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$$

❤ Projeção (π)

A operação projeção seleciona certas colunas da tabela e descarta as outras. Se estivermos interessados apenas em certos atributos de uma relação, usamos essa operação para projetar a relação apenas por esses atributos.



A projeção é representada pelo símbolo π. A lista de atributos desejada aparece subscrita a π e o argumento da relação é dado entre parênteses.

$$\pi_{<\text{lista atributos}>}(R)$$

O resultado da operação PROJEÇÃO tem apenas os atributos especificados em <lista atributos> na mesma ordem em que eles aparecem na lista. No exemplo abaixo, estamos listando último nome, primeiro nome e salário de cada funcionário:

$$\pi_{\text{Unome, Pnome, Salario}}(\text{FUNCIONARIO})$$

Eliminação de duplicatas: a projeção remove quaisquer tuplas duplicadas, de modo que o resultado é um conjunto de tuplas distintas.

A comutatividade **NÃO** é mantida na operação de projeção.

❤ **Renomeação (ρ ou ←)**

Operações em álgebra relacional podem ser escritas em uma única expressão ou aplicar uma operação de cada vez, e criar relações de resultado intermediário. Estas relações devem ser renomeadas:

$$\text{FUNCS\_DEPT5} \leftarrow \sigma_{\text{Dnr=5}}(\text{FUNCIONARIO})$$
$$\text{RESULTADO} \leftarrow \pi_{\text{Pnome, Unome, Salario}}(\text{FUNCS\_DEP5})$$

Às vezes, é mais simples desmembrar uma sequência complexa de operações do que escrever a expressão inteira. Também é possível renomear atributos nas relações intermediárias e de resultado:

$$\text{TEMP} \leftarrow \sigma_{\text{Dnr=5}}(\text{FUNCIONARIO})$$
$$R(\text{Primeiro\_nome, Ultimo\_nome, Salario}) \leftarrow \pi_{\text{Pnome, Unome, Salario}}(\text{TEMP})$$

A renomeação formal é indicada com ρ em qualquer uma das três formas abaixo:

$$\rho_{S(B1, B2, \ldots, Bn)}(R) \quad \text{ou} \quad \rho_{S}(R) \quad \text{ou} \quad \rho_{(B1, B2, \ldots, Bn)}(R)$$

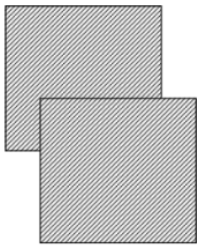
Em SQL, a renomeação é obtida usando AS.

OBS: No caso de ambiguidade de atributos, além da renomeação também pode-se utilizar a notação ponto para indicar a relação de cada atributo: Relação.nome

❤ **Operações de Conjunto**

União (R ∪ S): é uma relação que inclui todas as tuplas que estão em R ou em S ou em ambos. Tuplas duplicadas são eliminadas.

União

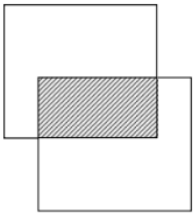


No exemplo abaixo, para recuperar os números de CPF de todos os funcionários que ou trabalham no departamento 5 ou supervisionam diretamente um funcionário que trabalha no departamento 5, podemos usar a união da seguinte forma:

$$\text{FUNCS\_DEP5} \leftarrow \sigma_{\text{Dnr=5}}(\text{FUNCIONARIO})$$
$$\text{RESULTADO1} \leftarrow \pi_{\text{Cpf}}(\text{FUNCS\_DEP5})$$
$$\text{RESULTADO2(Cpf)} \leftarrow \pi_{\text{Cpf\_supervisor}}(\text{FUNCS\_DEP5})$$
$$\text{RESULTADO} \leftarrow \text{RESULTADO1} \cup \text{RESULTADO2}$$

Interseção (R ∩ S): é uma relação que inclui todas as tuplas que estão tanto em R quanto em S.

Interseção

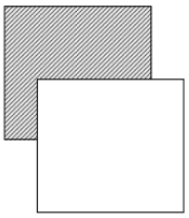


Exemplo: Listar os CPF's dos empregados que possuem dependentes e trabalham no departamento 5.

$$\text{Resultado1} \leftarrow \pi_{\text{CPF}}(\sigma_{\text{D\_Num=5}}(\text{Empregado}))$$
$$\text{Resultado2(CPF)} \leftarrow \pi_{\text{E\_CPF}}(\text{Dependente})$$
$$\text{Resultado} \leftarrow \text{Resultado1} \cap \text{Resultado2}$$

Diferença (R – S): é uma relação que inclui todas as tuplas que estão em R, mas não estão em S.

Diferença



Ex: Encontrar todos os CPF's dos empregados que não possuem dependentes.

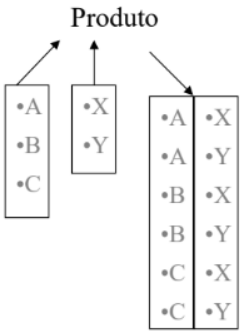
$$\text{Resultado1} \leftarrow \pi_{\text{CPF}}(\text{Empregado})$$
$$\text{Resultado2(CPF)} \leftarrow \pi_{\text{E\_CPF}}(\text{Dependente})$$
$$\text{Resultado} \leftarrow \text{Resultado1} - \text{Resultado2}$$

Quando essas operações são adaptadas aos BDs relacionais, as duas relações sobre as quais qualquer uma dessas operações são aplicadas precisam ter o mesmo tipo de tuplas; essa condição é chamada de compatibilidade de união ou **compatibilidade de tipo**.

- Duas relações são consideradas **compatíveis** se as duas duas relações têm o mesmo número de atributos e cada par correspondente de atributos tem o mesmo domínio.
- A União e a Interseção são comutativas
- A diferença **NÃO** é comutativa.

Produto Cartesiano (R X S): Também conhecido como produto cruzado ou junção cruzada, essa operação é usada para combinar (concatenar) as tuplas de duas relações. Apesar de ser uma operação de conjunto, as relações que participam de um produto cartesiano não precisam ser compatíveis.

O produto cartesiano cria tuplas com os atributos combinados de duas relações:



Essa operação aplicada isoladamente não tem significado. Ela é mais útil quando seguida por uma seleção que combina valores de atributos vindos das relações componentes. Por exemplo, suponha que queiramos recuperar uma lista dos nomes dos dependentes de cada funcionária:

$$\text{FUNC\_MULHERES} \leftarrow \sigma_{\text{Sexo='F'}}(\text{FUNCIONARIO})$$
$$\text{FUNCNOMES} \leftarrow \pi_{\text{Pnome, Unome, Cpf}}(\text{FUNC\_MULHERES})$$
$$\text{FUNC\_DEPENDENTES} \leftarrow \text{FUNCNOMES} \times \text{DEPENDENTE}$$
$$\text{DEPENDENTE\_PARTIC} \leftarrow \sigma_{\text{Cpf=Fcpf}}(\text{FUNC\_DEPENDENTES})$$
$$\text{RESULTADO} \leftarrow \pi_{\text{Pnome, Unome, Nome\_dependente}}(\text{DEPENDENTE\_PARTIC})$$

❤ Junção

A operação junção, indicada por ⋈, é usada para combinar tuplas relacionadas de duas relações em uma única tupla 'maior'. Essa operação é muito importante para qualquer BD relacional com mais de uma relação única, porque nos permite processar relacionamentos entre as relações. Forma geral da operação:

$$R \bowtie_{\text{<condição junção>}} S$$

A condição de junção é especificada sobre atributos das duas relações R e S e é avaliada para cada combinação de tuplas. Assim, o resultado de uma junção é uma relação que combina uma tupla de R e uma de S sempre que a combinação satisfaz a condição de junção. Uma condição de junção tem a forma:

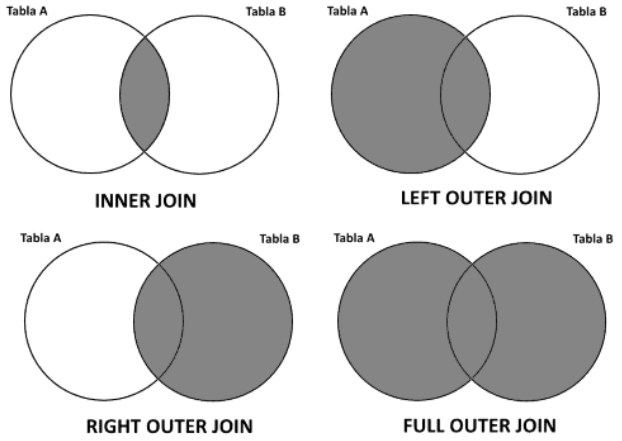
<condição> **AND** <condição> **AND**...
AND <condição>

Exemplo de junção: Recuperar os nomes do gerente de cada departamento.

$$\text{Ger\_Dep} \leftarrow \text{Departamento} \blacktriangleright\!\blacktriangleleft_{\text{CPFGer=CPF}} \text{Empregado}$$
$$\text{Resultado} \leftarrow \pi_{\text{Departamento.nome, Sobrenome, empregado.nome}}(\text{Ger\_Dep})$$

Tipos de junção

As junções podem ser internas (inner) ou externas (outer).



Uma operação de junção com uma condição de junção geral (=, <, ≤, >, ≥, ≠) é chamada de junção **Theta**.

Equijunção: junção mais comum que utiliza apenas o operador de igualdade (=).

No resultado de um equijunção, sempre temos um ou mais pares de atributos que possuem valores idênticos em cada tupla, porque a condição de junção de igualdade especificada sobre esses dois atributos requer que os valores sejam idênticos. Como cada par de atributos iguais é desnecessário, a junção natural foi criada.

Junção natural: é representada por * e requer que os dois atributos de junção (ou cada par de atributos de junção) tenham o

mesmo nome nas duas relações, o que resulta na remoção dos atributos desnecessários da equijunção.

Uma definição geral, mas não padrão para a junção natural é:

$$Q \leftarrow R *_{(\text{<lista1>}), (\text{<lista2>})} S$$

Suponha que queiramos combinar cada tupla PROJETO com a tupla DEPARTAMENTO que controla o projeto. No exemplo a seguir, primeiro renomeamos o atributo Dnumero de DEPARTAMENTO para Dnum (de modo que ele tenha o mesmo nome do atributo Dnum em PROJETO) e depois aplicamos a junção natural:

$$\text{PROJETO\_DEP} \leftarrow \text{PROJETO} * \rho_{(\text{Dnome, Dnum, Cpf\_gerente, Data\_inicio\_gerente})}(\text{DEPARTAMENTO})$$

O atributo Dnum é chamado de **atributo de junção** pois é o único atributo com o mesmo nome nas duas relações.

OBS: Uma operação junção pode ser especificada como um produto cartesiano seguido por uma operação seleção.

$$R \bowtie_{\text{<condição>}} S \equiv \sigma_{\text{<condição>}}(R \times S)$$

❤ Divisão

A operação de divisão, representada pelo símbolo ÷, é usada para resolver consultas do tipo “para todos”. A divisão é aplicada em duas relações R(Z) ÷ S(X), em que X ⊆ Z. Seja T(Y) a relação resultante da divisão:

- Y = Z – X
- Y é o conjunto de atributos de R que não estão em S

Exemplo: Recuperar os nomes dos empregados que trabalham em todos os projetos nos quais ’John Smith’ trabalha.

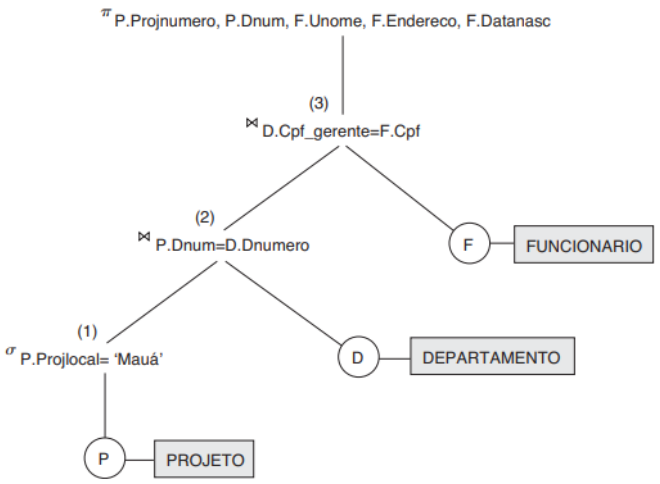
$$\text{Smith} \leftarrow \sigma_{\text{Nome="John" E Sobrenome="Smith"}}(\text{Empregado})$$
$$\text{R1} \leftarrow \pi_{\text{NumP}}(\text{Trabalha\_Em} \blacktriangleright\!\blacktriangleleft_{\text{CPFE=CPF}} \text{Smith})$$
$$\text{R2} \leftarrow \pi_{\text{CPFE, NumP}}(\text{Trabalha\_Em})$$
$$\text{R3(CPF)} \leftarrow \text{R2} \div \text{R1}$$
$$\text{Resultado} \leftarrow \pi_{\text{Nome, Sobrenome}}(\text{R3} * \text{Empregado})$$

A operação divisão pode ser expressa como uma sequência de operações π, x e – da seguinte forma:

$$T1 \leftarrow \pi_{Y}(R)$$
$$T2 \leftarrow \pi_{Y}((S \times T1) - R)$$
$$T \leftarrow T1 - T2$$

❤ Árvores de Consulta

Uma árvore de consulta é uma estrutura de dados em árvore que corresponde a uma expressão da álgebra relacional. Ela representa as relações de entrada da consulta como nós folha da árvore, e representa as operações da álgebra relacional como nós internos.



❤ Funções de Agregação

Na álgebra relacional básica, também é possível especificar funções de agregação matemáticas sobre coleções de valores do BD. Essas funções são usadas em consultas estatísticas simples que resumem informações das tuplas BD.

- Funções comuns: soma, média, máximo, mínimo e conta.

Outro tipo comum de solicitação envolve agrupar as tuplas em uma relação pelo valor de alguns de seus atributos e, depois, aplicar uma função de agregação independentemente para cada grupo. Para definir uma operação função agregada, é usado o símbolo ℑ ou Y. Especifica-se os tipos de solicitações de seguinte forma:

<atributos agrupamento> ℑ <lista funções> (R)

Onde <atributos agrupamento> é uma lista de atributos da relação especificada em R, e <lista funções> é uma lista de pares (<função> <atributo>).

Em cada par desse tipo, <função> é uma das funções permitidas (soma, média, máximo, mínimo, conta) e <atributo> é um atributo da relação especificada por R.

Exemplo: Recuperar cada número de departamento, o número de funcionários no departamento e seu salário médio.

d_num ℑ count Cpf, avg Salario (Empregado)

❤ Modificações no BD

Inserção: Para inserir dados em uma relação, é preciso especificar uma tupla ou escrever uma consulta que resulte em um conjunto de tuplas a inserir.

É representada por **r ← r ∪ E**, onde r é uma relação e E uma expressão em álgebra relacional. Uma inserção pode violar 4 tipos de restrições:

- **Domínio:** se o valor de um atributo não estiver contido no domínio correspondente.
- **Chave:** se um valor chave na nova tupla t já existir em uma outra tupla na relação r.
- **Integridade da entidade:** se a chave primária da nova tupla t for nula.
- **Integridade referencial:** se o valor de alguma chave estrangeira em t se referir a uma tupla que não exista na relação referenciada.

Projeto ← Projeto ∪ {("ProdutoJ", 15, "Stafford", 1)}

Exclusão: É expressa como uma consulta, mas, em vez de mostrar as tuplas ao usuário, remove as tuplas selecionadas do BD. É representada por **r ← r - E**. A operação excluir pode violar a **integridade referencial.**

Alicia(CPFE) ← π_{CPF} ($\sigma_{Nome = "Alicia" \text{ E } Sobrenome="Zelaya"}$ (empregado))

Dep_Alicia ← Alicia * Dependente

Dependente ← Dependente – Dep_Alicia

Atualização: É utilizada para alterar os valores de um ou mais atributos numa tupla de alguma relação.

- Para atualizar atributos que não sejam chave, é preciso verificar se o novo valor pertence ao domínio.
- Modificar uma PK é semelhante a excluir uma tupla e inserir outra: vale as restrições já citadas.
- Modificar uma FK: garantir que o novo valor se refira a uma tupla existente na relação referencial.

Exemplo: Suponha que todos os empregados recebam um aumento de salário de 3%.

❤ Resumo dos símbolos

- σ : seleção
- π : projeção
- X : produto cartesiano
- ρ ou ← : renomeação
- ⋈ : junção
- ∪ : união
- ∩ : interseção
- – : diferença
- ÷ : divisão
- ℑ ou Y : funções de agregação

Normalização

Existem quatro diretrizes informais que podem ser usadas como medidas para determinar a qualidade de projeto do esquema da relação:

1. Garantir que a semântica dos atributos seja clara no esquema.
2. Reduzir a informação redundante nas tuplas.
3. Reduzir os valores NULL nas tuplas.
4. Reprovar a possibilidade de gerar tuplas falsas.

❤ Semântica

Sempre que agrupamos atributos para formar um esquema de relação, consideramos que aqueles atributos pertencentes a uma relação têm certo significado no mundo real e uma interpretação apropriada associada a eles.

A semântica de uma relação refere-se ao seu significado resultante da interpretação dos valores de atributo em uma tupla. É como os valores dos atributos de uma tupla se relacionam uns com os outros.

Diretriz 1: Projete um esquema de relação de modo que seja fácil explicar seu significado.

❤ Informações redundantes

Um objetivo do projeto de esquema é minimizar o espaço de armazenamento usado pelas relações.

O armazenamento de informações repetidas leva a um problema adicional conhecido como anomalias de atualização, que podem ser classificadas em:

- **Anomalia de inserção:** não ser possível adicionar um dado sem que outro dado esteja disponível. ocorre, por exemplo, numa tabela tentar adicionar um departamento novo sem nenhum funcionário.
- **Anomalia de exclusão:** acontece quando a exclusão de um dado pode levar à perda de informações adicionais que deveriam ser preservadas. **Anomalia de modificação:** Ocorre quando, ao atualizar um dado, é necessário realizar múltiplas atualizações em diferentes locais da tabela, o que pode levar a inconsistências se alguma dessas atualizações for esquecida.

Diretriz 2: Projete os esquemas de relação da base de modo que nenhuma anomalia de inserção, exclusão ou modificação esteja presente nas relações.

❤ **Valores nulos**

Se muitos dos atributos não se aplicarem a todas as tuplas na relação, acabamos com muitos NULLs nessas tuplas. Isso pode desperdiçar armazenamento e ocasionar problemas com o conhecimento do significado dos atributos.

NULLs podem ter vários significados: atributo não se aplica a tupla; valor desconhecido; valor conhecido, mas ausente. Ter a mesma representação para todos os NULLs compromete os diferentes significados que eles podem ter.

Diretriz 3: Ao máximo possível, evite colocar atributos em uma relação da base cujos valores podem ser NULL com frequência. Se os NULLs forem inevitáveis, garanta que eles se apliquem apenas em casos excepcionais, e não à maioria das tuplas na relação.

❤ **Tuplas ilegítimas/falsas**

A criação de tuplas com informações inválidas acontece quando a aplicação de um JOIN em duas relações não obtém as informações originais.

Diretriz 4: Projete esquemas de relação de modo que possam ser unidos com condições de igualdade sobre os atributos que são pares relacionados corretamente (PK, FK) de um modo que garanta que nenhuma tupla falsa será gerada.

❤ **Dependência funcional (DF)**

Uma dependência funcional é uma restrição entre dois conjuntos de atributos do BD. É representada por A → B, lê-se:

- A funcionalmente determina B;
- B é funcionalmente dependente de A;
- B é função de A.

P/ cada valor de A só existe um valor de B.

Exemplo: Considerando a relação Emp_Proj (CPF, NumeroP, Horas, NomeP, NomeE, LocalizacaoP), pela semântica dos atributos e da relação, as seguintes DFs devem ser mantidas:

- CPF → NomeE
- NumeroP → {NomeP, LocalizacaoP}
- {CPF, NumeroP} → Horas

A DF é uma propriedade semântica ou de significado dos atributos. Em outras palavras, a **dependência funcional** se refere a atributos não-chaves que dependem de outros atributos para fazerem sentido em uma relação.

Dependência funcional total: Em uma relação com PK composta, o atributo não-chave depende de todas as partes da PK. Se remover alguma das partes, a dependência não se mantém. Ex: codPedido (PK), codProduto(PK) e qtdProduto.

Dependência funcional parcial: Em uma relação com PK composta, o atributo não-chave depende de apenas uma parte da PK. Se remover alguma das partes, a dependência se mantém. Ex: idAluno (PK), codDisciplina (PK), nomeDisciplina.

Dependência funcional transitiva: Ocorre quando um atributo depende de outro que não faz parte da chave primária da relação a qual ele pertence. Geralmente, indica que esse atributo não deveria estar nessa relação/tabela. Ex: numPedido (PK), codVendedor(FK), nomeVendedor.

Dependência funcional multivalorada: Ocorre quando, para cada valor de atributo de A, existe um conjunto de valores de atributos B e C que estão associados a ele, mas são independentes entre si. Ex: **modelo** (A), ano (B) e cor (C) de um carro.

A ideia da normalização é agrupar numa tabela somente dois conjuntos de atributos: A e B, com A → B. A é então a **chave da tabela** ou **determinante**, e B é **complemento da chave**.

Forma Normal (FN)

A normalização é um processo formal passo a passo que examina os atributos de uma entidade com base em suas dependências funcionais e chaves primárias, com o objetivo de minimizar redundância e de evitar anomalias de inclusão, exclusão e alteração.

As FN's são conjuntos de restrições nos quais os dados devem satisfazê-las. Na normalização, os esquemas de relação são **decompostos** em esquemas menores, atendendo duas condições:

- Propriedade de junção não aditiva ou junção sem perdas (sem tuplas falsas). Deve ser alcançada a todo custo.
- Propriedade de preservação de dependência. Desejável, mas pode ser sacrificada.

A normalização completa dos dados é feita seguindo as restrições das formas normais existentes. A passagem de uma FN para outra é feita tendo como base o resultado da FN anterior.

O ideal é que o projeto do BD relacional alcance a FNBC ou a 3FN para cada tabela.

❤ Primeira Forma Normal (1FN)

A 1FN foi definida para reprovar atributos multivalorados, atributos compostos e suas combinações. Os únicos valores de atributos que são permitidos são valores atômicos individuais. A 1FN também não permite relações aninhadas (uma tabela dentro de outra tabela).

Passos:

- Identificar a chave primária da entidade;
- Identificar o grupo repetitivo e removê-lo da entidade;
- Criar uma nova entidade com a chave primária da entidade anterior e o grupo repetitivo.

Uma tabela está na 1FN quando:

- Somente possui valores atômicos (indivisíveis).
- Não há grupos de atributos repetidos.
- Existe uma chave primária.
- Não possui atributos multivalorados e relações aninhadas.

Exemplo e solução em 1FN:

CPF	Nome	Endereço	Habilidade
8795835	Fulano da Silva	Rua X, 13	{Futebol, Voleibol, Basquete, Atletismo, Tênis}

CPF	Nome	Endereço
8795835	Fulano da Silva	Rua X, 13

CPF	Esporte
8795835	Futebol
8795835	Voleibol
8795835	Basquetebol
8795835	Atletismo
8795835	Tênis

Uma tabela em 1FN **NÃO** evita anomalias.

❤ Segunda Forma Normal (2FN)

A 2FN é baseada no conceito de **dependência funcional total**. Uma tabela está na 2FN, se ela estiver na 1FN e não possuir campos que são funcionalmente dependentes de parte da chave. Ou seja, cada um dos atributos não-chaves são totalmente dependentes da PK inteira.

A 2FN serve para que cada coluna se relacione diretamente com sua chave primária e não dependa de outra coluna. Se houver informações que pertençam a registros múltiplos, deve-se criar uma nova tabela para evitar a repetição de dados.

Cada atributo de uma tabela deve depender da PK. Se a PK possuir um único atributo, não precisa fazer o teste da 2FN.

Passos:

- Identificar os atributos que não são funcionalmente dependentes de toda a chave primária.
- Remover da entidade todos esses atributos identificados e criar uma nova entidade com eles.
- A chave primária da nova entidade será o atributo do qual os atributos removidos são funcionalmente dependentes.

Uma tabela está na 2FN quando:

- Está na 1FN.
- Todos os atributos não-chaves são DF de todas as partes da PK.
- Não existem dependências parciais.

Resumo: Deve-se criar uma nova relação para cada PK ou combinação de atributos que forem determinantes em uma dependência funcional.

Esse atributo será a PK da nova tabela. Mova os atributos não-chave dependentes dessa PK para a nova tabela. Exemplo:

Matrículas			
ID_Aluno	Cod_Disciplina	Nome_Disciplina	Data_Inicio
1	101	Banco de Dados	01/02/2024
2	102	Algoritmos 1	05/06/2023
3	105	Redes 2	05/06/2023
4	108	Teste de SW	01/02/2024

Cod_Disciplina → Nome_Disciplina, mas a chave primária da tabela Matriculas é composta por ID_Aluno e Cod_Disciplina. Há dependência parcial. Solução em 2FN:

Matrículas		
ID_Aluno	Cod_Disciplina	Data_Inicio
1	101	01/02/2024
2	102	05/06/2023
3	105	05/06/2023
4	108	01/02/2024

Disciplina	
Cod_Disciplina	Nome_Disciplina
101	Banco de Dados
102	Algoritmos 1
105	Redes 2
108	Teste de SW

Uma tabela em 2FN **NÃO** evita anomalias totalmente.

❤ Terceira Forma Normal (3FN)

A 3FN é baseada no conceito de **dependência transitiva**. A relação não deve ter um atributo não-chave determinado funcionalmente por outro atributo não-chave (ou conjunto). Ou seja, não deve haver **dependência transitiva** de um atributo não-chave sobre a PK.

Assim, a 3FN cria tabelas adicionais para separar campos que não dependem da chave primária, mas estão relacionados com outros campos. Se um atributo não chave depender de outro atributo não

chave, deve ser eliminado da tabela e colocado em uma nova tabela separada.

Uma tabela está na 3FN se:

- Está na 2FN.
- Não existem dependências transitivas.

Passos: Para cada atributo não-chave cria-se uma nova tabela, onde esse atributo será a PK. Mova então todos os atributos que são dependentes funcionalmente dessa PK. O atributo que se tornou PK na nova relação fica também na tabela original e servirá como uma FK para associar as duas relações.

Exemplo:

Vendas				
Nota_Fiscal	Cod_Vendedor	Nome_Vendedor	Cod_Produto	Qtde_vendida
15326	002	Leila	132	10
15327	006	Ana	153	12
15328	002	Leila	143	11
15329	009	Fábio	132	9
15330	007	Renato	153	12

Cod_Vendedor → Nome_Vendedor, mas Cod_Vendedor não é PK da tabela vendas. Há dependência transitiva. Solução 3FN:

Vendas			
Nota_Fiscal	Cod_Vendedor	Cod_Produto	Qtde_vendida
15326	002	132	10
15327	006	153	12
15328	002	143	11
15329	009	132	9
15330	007	153	12
PK	FK		

Vendedor	
Cod_Vendedor	Nome_Vendedor
002	Leila
006	Ana
002	Leila
009	Fábio
007	Renato
PK	

Uma tabela em 3FN evita algumas anomalias.

❤ Resumo da Normalização

Tabela não normalizada → Remover atributos multivaloradas e compostos → Remover dependências parciais → Remover dependências transitivas

❤ Forma Normal de Boyce-Codd (FNBC)

A BCNF é uma extensão da 3FN e foi desenvolvida para lidar com anomalias de atualização que ainda podem existir após um banco de dados estar na 3FN.

Cada relação na FNBC está na 3FN, mas uma relação 3FN não está necessariamente na FNBC.

Definição: Um esquema de relação está na BCNF se, para cada uma de suas dependências funcionais (X → Y), X é uma superchave.