

# **Bases de Dados**

---

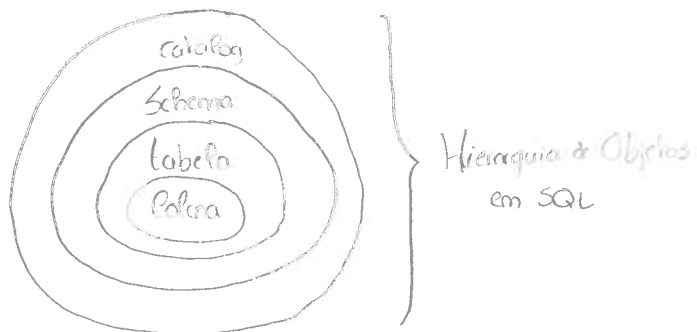
Resumos  
2016/2017

Carolina Albuquerque | 80038

# Linguagem SQL - Structured Query Language

Linguagem para definir, manipular e questionar uma base de dados relacional

Sublinguagens { DDL - Data Definition Language - a que damos  
DDL - Data Manipulation Language



## Linguagem DDL

create database companhia; -- cria uma base de dados como nome companhia  
drop database companhia; -- apaga " " " "

create schema rent-a-car; -- cria um schema: agrupa tabelas e outros elementos pertencentes à mesma aplicação  
drop schema rent-a-car; -- apaga o schema

Tipos de Dados {  
- numbers  
- chars, strings  
- data, time  
- binary objects

! Ver tabela slides

! Nota: Usar tipos de dados compatíveis com standard - aumenta a probabilidade de sucesso

criar tabelas + atributos } ver slides


## Restrições de Integridade em SQL

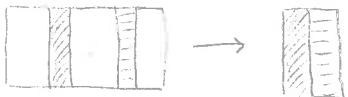
check(p) - impor uma regra a um atributo  
not null - atributo não pode ser null  
primary key - define chave primária  
unique - chaves candidatas não primárias  
foreign key - define chave estrangeira

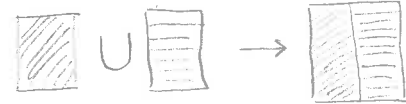
Podem ser de tabela ou de coluna  
ver slides com exemplos

## Álgebra Relacional

### Operações Básicas:

① Seleção   $\sigma_{\text{selecção condição}}(R)$   
relação

② Projecção   $\pi_{\text{lista atributos}}(R)$   
relação

③ União  SQL: UNION(ALL)  
INTERSECT(ALL)  
EXCEPT(ALL)

④ Diferença 

⑤ Produto Cartesiano = Cross join

⑥ Renomeação(p)  $TEMP \leftarrow \sigma_{\text{atributo}}(Employee)$   
atribuição

## Junção $\theta$ (Theta Join)


$R \bowtie_{c,s} S$  } Notação

Pode ser visto como resultado das operações

$R_3 \leftarrow R_1 \times R_2$  (produto cartesiano)

$\sigma_c(R_3)$  (seleção com condição  $c$ )

$c$  é join condition

 Nota: Ver o resto de algebra relacional nos slides

## Linguagem SQL View

- > Pode ser usada como fonte de dados (idêntica a uma tabela normal) num conjunto de operações
- > Usada na apresentação de dados
- > Usada para questões de segurança de dados
- > Ferramenta de estruturação de queries mais complexas
- > As vistas podem ser consideradas como tabelas virtuais: têm um conjunto de definições e armazenam fisicamente os dados
- > Uma vista tem um conjunto de definições criadas sobre tabelas ou outras vistas e não armazena fisicamente os dados

Relação Virtual - derivada de relações base  
- permite manipular os dados da view

 Nota: É basicamente uma cópia da seleção que temos

## Exercícios Álgebra Relacional

a)  $\pi$  Ssn, Fname, Iinit, Lname, Pno  
(employee  $\bowtie$  Ssn = Essn works-on)

c)  $(\gamma$  Pno; SUM(Hours)  $\rightarrow$  Hours (works-on))  $\bowtie$   
(project  $\bowtie$  Pnumber = Pno works-on)

d)

$\sigma$  Pname = "Aveiro Digital" (works-on  $\bowtie$  Pno = Pnumber Project)

"

## Exercícios Álgebra Relacional - Slides:

① Nome dos fármacos que nunca foram prescritos  
 $\pi$  name  
 $(\sigma_{\text{farmaco} = \text{null}}(\text{Prescribe} \bowtie \text{farmaco} = \text{código Farmaco}))$

② Número de fármacos prescritos em cada consulta  
 $\pi$  medico, consulta, num-farm = count(farmaco) Prescribe  
ou  
medico, consulta  $\gamma$  count(farmaco) (Prescribe)

③ Para cada médico, a quantidade média de fármacos por consulta  
 $\pi$  medico, consulta, avg-farmaco (count(farmaco) (Prescribe))

## Normalização

> Prevenção de Informação: Todos os conceitos capturados pelo desenho conceitual que são mais tarde mapeados para o desenho lógico.

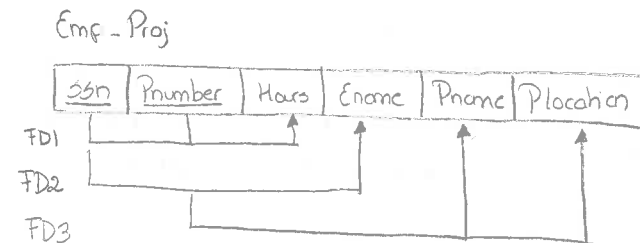
> Minimiza a redundância de dados: Minimizar o armazenamento de dados duplicados em relações distintas, reduzindo a necessidade de múltiplos updates e consequentemente problemas de consistência entre múltiplas cópias da mesma informação.

- Critérios Informais
- Redução dos NULLs nos tuplos  
A existência de muitos nulls nos tuplos é desperdício de espaço e dificulta a interpretação do seu sentido desses atributos. Solução: Criar uma nova relação para esses atributos
  - Clareza da semântica dos atributos da relação
  - Redundância da informação no tuplo
  - Junção de relações baseada em PK e FK

## Dependência Funcional (DF)

- > Formalismo de análise de esquemas relacionais
- > Permite descrever restrições dos atributos que os tuplos devem respeitar em todo o momento (invariantes).
- > Permite detectar e descrever problemas com precisão
- > Uma DF é uma propriedade do esquema relação  $R$  que não pode ser inferido de uma qualquer instância de  $R$ , ou seja  $r(R)$

## Exemplo:



Ssn  $\rightarrow$  Ename

Prnumber  $\rightarrow$  { Pname, Plocation }

{ Ssn, Prnumber }  $\rightarrow$  Hours

- > Ssn determina de forma única o nome do funcionário
- > Prnumber do projeto determina de forma única o nome e a localização do projeto

## Tipos DF

- Parcial: atributo depende de parte dos atributos que compõem a chave da relação
- Total: atributo depende de toda a chave da relação
- Transitiva: atributo que não faz parte da relação depende de um atributo que também não faz parte da chave da relação

## Processo de Normalização

↳ Formas Normais:

- > Conjuntos de testes (condições) para validação de cada forma
- > Cada forma superior tem menos DF que a anterior
- > Primeira (1FN) Segunda (2FN) e Terceira (3FN)



A 3FN satisfaz as condições da 2FN e está a dos 1FN

- > 3FN Passou a BCNF (mais restrita)
- > Posteriormente, propuseram 4FN e 5FN

## 1ª Forma Normal (1FN)

Uma relação está na 1FN se:

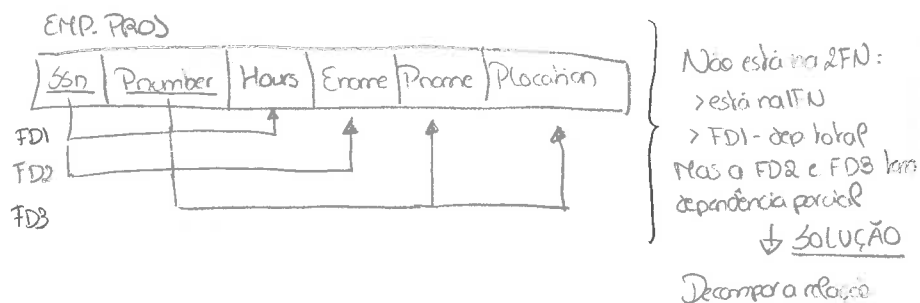
- > Cada atributo contém apenas valores atômicos (simples e indivisíveis) não são permitidos atributos compostos ou multi-valor
- > Não suporta relações dentro de relações - não é possível utilizar uma relação como valor de um atributo de um tuplo

⚠ Nota: Ver exemplos nos slides

## 2ª Forma Normal (2FN)

Uma relação está na 2FN se estiver na 1FN e:

- > Os atributos que não são chave dependem na totalidade da chave (não existem dependência parcial)



## 3ª Forma Normal (3FN)

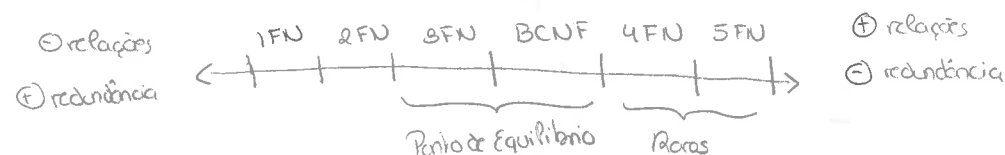
Uma relação está na 3FN se está na 2FN e

- > Não existem dependências funcionais entre atributos não chaves, não existem dependências transitivas

⚠ É a forma que normalmente termina o processo de normalização

## BCNF (Boyce-Codd Normal Form)

- > Todos os atributos são funcionalmente dependentes da chave relação de toda a chave e de nada mais



⚠ Nota: Geralmente uma relação na BCNF está na 4FN e 5FN

## Indices (Indexes)

- > Estruturas dados que oferecem uma segunda forma de acesso aos dados

↓ Consequências

- ✓ Reduz o tempo de consulta
- ✗ Pode aumentar o volumen de dados armazenados, e o tempo das inserções overload

- > É possível indexar qualquer atributo da relação
- > Prior múltiplos índices (sobre atributos distintos)
- > Prior índices com vários atributos

Atributos Indexados - Index Key

Os índices têm um ponteiro para a sua localização

- Não Denso - início da página (bloco)
- Denso - Offset do próprio tuplo da página

### Tipos Genéricos

#### Single-Level Ordered

- > Estruturas de um único nível que indexam um atributo da relação
- > Índices são ordenados o que permite uma pesquisa binária sobre o atributo
- Analogia: Índice Puro (palavra - página)

> Complexidade pesquisa:

$$Pog_p b_i (b_i - \text{index blocks})$$

#### Multi-Level

- > Vários níveis de indexação passando a complexidade para:  
 $Pog_p b_i \quad fo - fan out > 2$
- > São implementados com estruturas em árvores balanceadas (B-Tree)

### Tipos: ① Primary Index

- Indexa um atributo chave da relação (não se repete)
- Tuplos armazenados em blocos de tamanho fixo

### ② Clustered Index

- Indexa um atributo que pode ter valores duplicados
- Atributos estão agrupados

### ③ Secondary Index

- Indexa outros atributos (chave condicional ou não chave)
- Podemos ter vários índices deste tipo



Só podemos ter um Primary ou um Clustered

## Índices - SQL Server

### Clustered

- Os nós folhas contêm os próprios dados da relação
- Tabela ordenada pelo próprio índice
- Exemplo: Agenda Contactos

### Non-clustered

- Índices apontam para a tabela base
- Podemos ter vários numa relação
- Exemplo: Índice no fim de um livro

Ambos implementados com B-Trees

Exemplos: ① -- Criar um índice para o atributo Prone

```
CREATE INDEX idxProne ON Project (Prone);
```

② -- Criar um índice multi-atributo

```
CREATE INDEX idxEmpName ON Employee (Fname, Iinitial, Lname)
```

③ -- Eliminar índice

```
DROP INDEX idxProne;
```

## Programação SQL

Batch - grupo de uma ou mais instruções SQL que constituem uma unidade lógica

Script - ficheiro texto contendo uma ou mais batches delimitadas por GO

Cursor - Ferramenta que permite percorrer sequencialmente os tuplos retornados por determinada consulta (SELECT)

## Stored Procedures

- > Batch armazenado com nome
- > Procedimentos são guardados em memória cache na 1ª vez que são executados → Execução ① rápida
- > Procedimento pode:
  - ter parâmetros de entrada
  - ter valor de retorno
  - devolver conjunto registros

Exemplos ① -- Devolver um conjunto de registros

```
CREATE PROCEDURE dbo.CategoryList
AS
SELECT ProductCategoryName, ProductDescription
FROM dbo.ProductCategory
GO
```

② -- Com parâmetros de entrada

```
CREATE PROC Department-Members @DepName varchar(50)
AS
SELECT DepName, COUNT(Emp-ID) Number of Member
FROM Departments D, Employees E
WHERE D.Dep-ID = E.Dep-ID and DepName = @DepName
GROUP BY DepName
GO
```

Tipos SP {

- System: podem ser utilizadas em qualquer base de dados
- Local: definidas numa base de dados local

### Execução:

EXEC dbo.CategoryList;

EXEC Department-Members 'Accounting'

EXEC GetTopProducts @End.ID = 10, @Start.ID = 1

## Vantagens:

- > Melhorias no desempenho eliminando a necessidade de múltiplas transmissões de informação através da rede
- > Segurança as operações de inserção, alteração, eliminação e consulta podem ser executadas a partir de um SP → aplicações deixam de ter necessidade de conhecimento completo da BD.
- > Separação Cliente-Servidor facilitam a identificação entre tarefas que devem ser executadas pelo lado do cliente e do servidor

## UDF (User Defined Functions)

- > Têm os benefícios que as stored procedures (são compiladas e otimizadas)
- > Podem ser usadas para incorporar lógica complexa dentro de uma consulta
- > Têm os benefícios que as vistas por poderem ser utilizadas como fonte de dados
- > Aceitam parâmetros (ao contrário das vistas)

- Escalares: aceitam múltiplos parâmetros retornam um único valor
- In line-Tablevalued: similares às vistas (ambas aceitam wrappers para construtores SELECT têm as ① vantagens das vistas ① suportam parâmetros de entrada)
- Multi-Statement Tablevalued: combina as capacidades dos tipos acima cria uma table variable, introduz-lhe linhas e retorna-a

! Nota: usamos no projeto

### ! Nota: Limitações UDFs escalares

- > Determinísticas (os mesmos parâmetros de entrada produzem o mesmo valor de retorno)
- > Não são permitidos updates à base de dados ou invocação do comando DECC
- > Não permite try...catch ou RaisingError
- > Recursividade limitada a 32 níveis

> Os 3 tipos de UDF podem ser definidos como Schema Binding

### Trigger

- > Tipo especial de stored procedure que é executado em determinadas circunstâncias associadas à manipulação de dados
- > Quando ocorre uma das ações previstas, os triggers são "disparados" (exec.)

SQL Server } - Triggers DML  
                  } - Triggers DDL

Exemplos de uso: ① Manutenção de dados duplicados e desnecessários

② Restrições de colunas complexas

③ Integridade referencial em cascata

④ Predefinições complexas

⑤ Integridade referencial em bancos de dados

DML Triggers { - Instead of: apenas um por tabela, não é executada a ação  
                  } - After: é possível ter várias triggers after por tabela, validação de dados / efetuar auditorias nos dados / atualizar campos calculados

### Instruções NÃO Permitidas num trigger

- CREATE, ALTER ou DROP database
- RECONFIGURE
- RESTORE database or log
- DISK RESIZE
- DISK INT

### Transação

- > unidade lógica de trabalho contendo um ou mais operações read(x) e write(x)

Exemplo: transferência bancária

```
BEGIN TRANSACTION
UPDATE authors SET au-name = upper (au-name)
WHERE au-name = 'white'
IF @@ROWCOUNT = 2
    COMMIT TRAN      ----- Sucesso
ELSE
    BEGIN
        PRINT 'A transaction needs to be rolled back'
        ROLLBACK TRAN      ----- Falha
    END
```

### Propriedades:

- > Atomicidade: operações ocorrem de uma forma indivisível (todas commit ou nenhuma falha)
- > Integridade: após operações mantém-se o estado integridade
- > Isolamento: Ilusão de ser transação única. Transações concorrentes não interferem entre si
- > Persistência: efeitos transação terminada com commit são permanentes e visíveis para outras transações

ACID



## Recuperação de Falhas

- Falha {
- ⊖ Grave: falha numa transação
  - ⊕ graves: perda total ou parcial da base de dados

↓ Como recuperar?

- Escalonamentos
- Backups
- Transaction Logging

### Escalonamentos

- ① Recuperável - se nenhuma transação  $T_i$  em  $E$  for concluída (committed) até que todas as outras transações que escrevem elementos lidos por  $T_i$  tenham sido concluídas
- ② Sem Abort em Cascata - O recuperável pode gerar aborts de transações em cascata

## Guia Prático 10

10.1 a) CREATE PROCEDURE RemoveFunc @Ssn-Func INT  
AS  
BEGIN  
DELETE FROM Works-on WHERE Essn = @Ssn-Func;  
DELETE FROM Dependent WHERE Essn = @Ssn-Func;  
UPDATE department SET Mgr-ssn = NULL, Mgr-startdate = NULL  
WHERE Mgr-ssn = @Ssn-Func  
UPDATE employee SET Super-ssn = NULL WHERE super-ssn =  
@Ssn-Func  
DELETE FROM employee WHERE Ssn = @Ssn-Func  
END;  
GO

b) CREATE PROCEDURE Mgr-Employee  
AS  
BEGIN  
SELECT employee.\* FROM department, employee  
WHERE Mgr-ssn = Ssn;  
SELECT TOP 1 Ssn, Datediff(year, Mgr-start-date, GetDate()) AS  
nanos  
FROM department, employee WHERE Mgr-ssn = Ssn  
GROUP BY nanos Desc,  
END;  
GO

c) CREATE TRIGGER MgrUnicoDep ON Department INSTEAD OF INSERT,  
UPDATE  
AS  
BEGIN  
DECLARE @MgrCount AS INT;  
SELECT @MgrCount = count(\*) FROM department, inserted  
WHERE department.Mgr-ssn = inserted.Mgr-ssn;  
IF @MgrCount > 0  
ROLLBACKTRAN;  
END  
GO  
ENABLE TRIGGER MgrUnicoDep ON Department ON

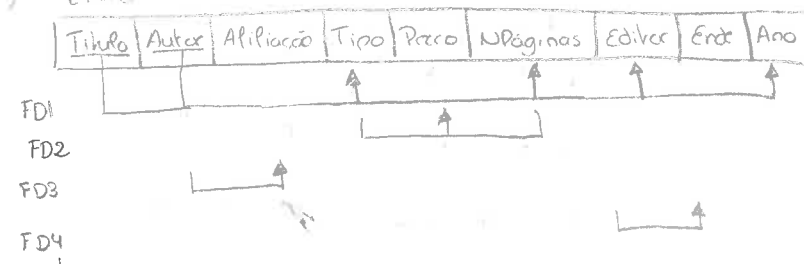
```

d) CREATE TRIGGER Vencimento_Ssn ON employee AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @MgsSalario INT;
    SELECT @Mgs.salario = employee.Salary FROM inserted, department,
    WHERE department.Dnumber = inserted.Dno
    AND employee.Ssn = Mgs.Ssn
    IF @MgsSalary <= (SELECT SALARY FROM inserted)
    BEGIN
        UPDATE Employee SET Salary = @MgsSalary - 1
        WHERE Ssn = (SELECT Ssn FROM inserted)
    END
END
GO
ENABLE TRIGGER Vencimento_Ssn ON employee

```

## Guia 9

9.1 a) Livro

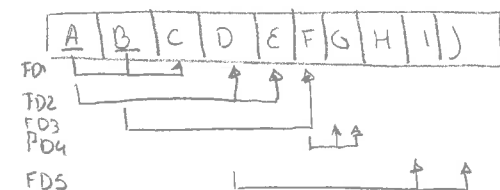


Está na 1FN porque existem dependências parciais

- b) FD1:  $\{Título, Autor\} \rightarrow \{Tipo, NPáginas, Editor, Ano\}$   
 FD2:  $\{Tipo, NPáginas\} \rightarrow \{Preço\}$   
 FD4:  $\{Editor\} \rightarrow \{Ano\}$   
 FD5:  $\{Autor\} \rightarrow \{Afiliação\}$   
 FD2:  $\{Tipo, NPáginas\} \rightarrow \{Preço\}$   
 FD3:  $\{Editor\} \rightarrow \{Ano\}$
- 2FN  
 3FN

9.2 a) A e B

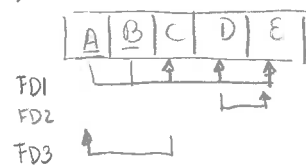
b) R



- $\{A, B\} \rightarrow \{C\}$   
 $\{A\} \rightarrow \{D, E\}$   
 $\{B\} \rightarrow \{E, F\}$   
 $\{F\} \rightarrow \{G, H\}$   
 $\{D\} \rightarrow \{I, J\}$
- 2FN  
 3FN

93 a) Chave: B, A

b) R



FD1:  $\{\underline{A}, \underline{B}\} \rightarrow C, D, E$   
FD2:  $\{\underline{D}\} \rightarrow E$   
FD3:  $\{\underline{C}\} \rightarrow A$  } 3FU

FD1:  $\{\underline{A}, \underline{B}\} \rightarrow C, D, E$   
FD2:  $\{\underline{D}\} \rightarrow E$  } BCNF