

## Introdução às bases de dados

O que é uma base de dados?

- Coleção de dados integrados
- Modela as características intrínsecas do universo que tenta abranger

O que é um SGBD?

- Sistema de Gestão de Base de Dados
- Software desenhado para armazenar e gerir grandes quantidades de dados e coordenar o acesso dos utilizadores

## Modelo de dados

- Coleção de construtores para descrever os dados a nível abstrato
- Esconde detalhes de armazenamento
- Define os dados que serão armazenados pelo SGBD
- Mais perto da estrutura de armazenamento do SGBD do que da visão do utilizador
- Descrição dos dados em termos de um modelo de dados é chamada de esquema (schema)

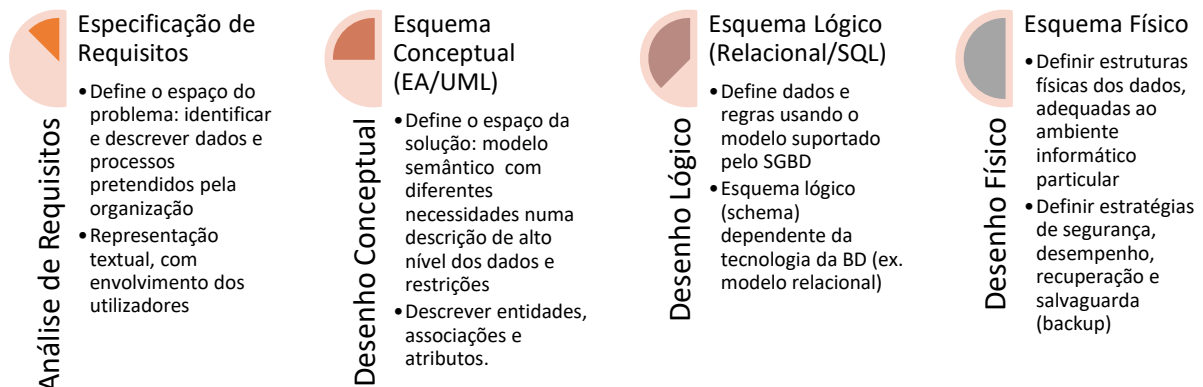
## Modelo mais usado na atualidade – Modelo Relacional

- Conceito fundamental é a Relação = tabela com colunas (atributos, campos) e linhas (registos)
- O esquema relacional – Descreve o nome das relações e suas colunas –  
Exemplo: Estudante (sid: string, name: string, login: string, age: integer, gpa: real)  
Cada linha na relação Estudante é um registo que descreve um aluno  
Cada linha segue o esquema da relação Estudante
- Inclui também regras de integridade, que são condições que os registos de uma relação têm de satisfazer
- Suportado diretamente pelos SGBDs relacionais – SGBDs têm comandos para criar tabelas e gerir registos de dados
- Nem sempre é o mais adequado – A primeira abordagem de modelação deve ser mais abstrata

## Modelo semântico dos dados

- Modelo mais rico com construtores mais ricos para descrever a realidade
- Mais perto da visão do utilizador
- No final é traduzido para um modelo de dados
- Modelo Entidade-Associação (EA) (Entity-Relationship (ER) model)
  - Descreve de uma forma gráfica as entidades e as relações entre ela
  - Ponto de partida da modelação de dados em BD

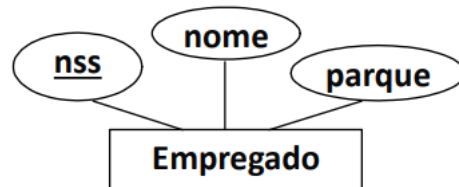
## Processo de Desenho de Bases de Dados



- Universo do Discurso – Fragmento do mundo real para o qual se pretende conceber o SI
- Estrutura de Conceitos – Conjunto de abstrações usadas para simbolizar entidades
- Modelo Semântico – Interpretação de um UoD através de uma estrutura de conceitos
- Entidade-Associação (EA)
  - Várias versões: várias simbologias, estrutura de conceitos semelhante (ex. Unified Modeling Language (UML))

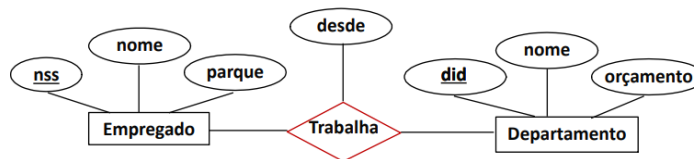
## Entidade

- Objeto do mundo real
- Conjunto de Entidades: coleção de entidades semelhantes
  - Partilham atributos
  - Partilham associações
- Chave - conjunto mínimo de atributos cujos valores identificam univocamente cada entidade do conjunto
  - Existem várias chaves candidatas
  - Uma é escolhida para ser chave primária
- Representado por um retângulo
- Atributos representados por elipses
- A chave primária está sublinhada



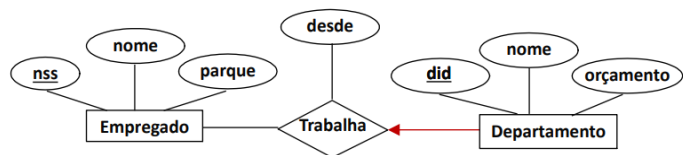
## Associação

- Relação entre duas ou mais entidades
- Conjunto de Associações: coleção de associações semelhantes
  - Relacionam os mesmos conjuntos de entidades:
- Binárias: dois conjuntos de entidades (não necessariamente distintos)
- Ternárias: três conjuntos de entidades
- Pode ter atributos descritivos – Com informação sobre a associação
- Representado por um losango



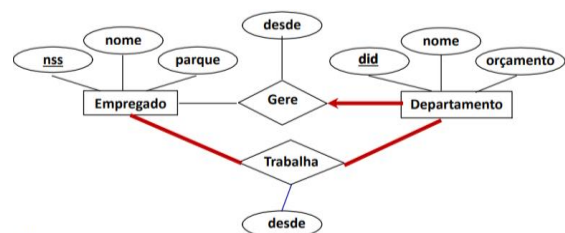
## Restrição de Chave

- Multiplicidade
- Máximo uma associação por cada entidade
- Representado por uma seta



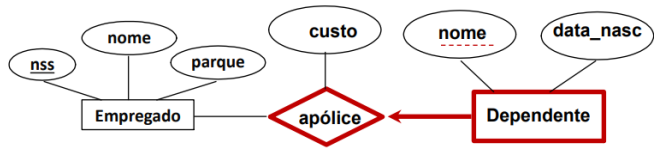
## Restrição de Participação

- No mínimo uma associação por cada entidade
- Pode existir participação total
  - Representada por uma linha a cheio
- Por omissão: assume-se participação parcial



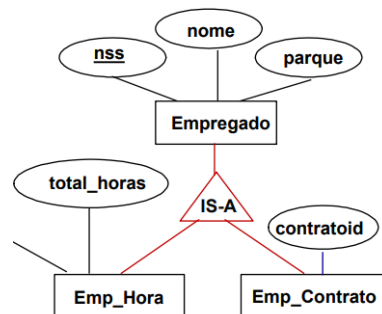
## Entidades Fracas

- Uma entidade fraca só pode ser identificada
  - Através dos seus atributos
  - E da chave primária de outra entidade - entidade forte
- As seguintes restrições têm de existir:
  - O conjunto de entidades fortes e o conjunto de entidades fracas devem participar numa associação um-para-muitos
  - Chamada de associação identificadora
- O conjunto de entidades fracas deve ter participação total na associação identificadora, representada por linhas a cheio
- A chave parcial está sublinhada a tracejado



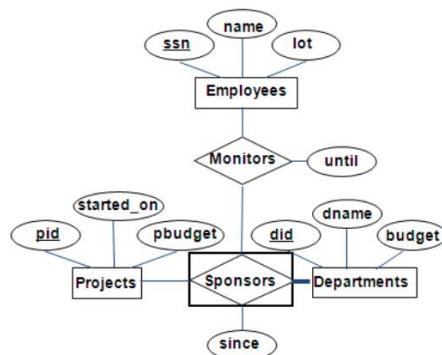
## Generalizações e Especializações

- Classifica um conjunto de entidades em subconjuntos
- Os atributos das super-entidades são herdados pelos subconjuntos de sub-entidades
- Especialização é o processo de identificação de subconjuntos de entidades que partilham características comuns: atributos ou associações
- Generalização consiste na identificação de algumas características comuns a vários conjuntos de entidades e representar essas características num novo conjunto de entidades
- Restrições de Sobreposição**
  - Restrições de sobreposição determinam se dois subconjuntos podem conter a mesma entidade
  - Sobrepostos ou disjuntos
  - Por omissão os subconjuntos são disjuntos
  - A restrição de sobreposição é definida com uma regra de integridade (RI)
- Restrições de Cobertura**
  - Restrições de cobertura determinam se todas as entidades estão representadas nos subconjuntos – Total ou parcial
  - Por omissão a cobertura é parcial
  - A restrição de cobertura total é definida com uma regra de integridade (RI)



## Agregação

- Agregação indica que um conjunto de associações está associado com um outro conjunto de associações



## Agregação vs. Ternária

- Numa associação ternária é obrigatório ter três entidades: – (a,b,c)
- Numa agregação, as associações são independentes: – ((a,b),c)

## Regras de Integridade Adicionais

- Modelo EA é expressivo, mas tem limitações
  - Poucas opções para o limite superior de participação numa associação
  - Restrição de chave limita a no máximo uma participação
  - Outros limites bem determinados não se representam graficamente
  - Não suporta relações bem conhecidas entre valores de atributos
- Restrições de integridade (RI) adicionais escritas em forma de texto
  - Frases curtas, com os mesmos termos usados no diagrama
  - Numeradas (RI-1, RI-2, ...)
  - Agrupadas e colocadas logo abaixo do diagrama

## Modelo Relacional

- Estrutura de dados
  - Base de dados é uma coleção de relações
  - Relação é uma tabela com linhas e colunas
  - Cada coluna tem valores de um domínio de dados
- Operadores relacionais
  - Para gestão de tabelas e outras estruturas
  - Para inserção, remoção e pesquisa de dados
- Regras de Integridade
  - Para garantia da coerência dos dados
- Relações são tabelas com linhas e colunas
  - colunas = atributos = campos
  - linhas = registos = entidades = tuplos
- Esquema de relação inclui
  - Nome da relação, Nome e domínio de dados de cada coluna

Ex: Students (sid: integer, name: string, login: string, age: integer, gpa: real)

## Propriedades das Relações

- Restrições de domínio
  - Especificam o tipo de dados de cada atributo
  - Todas as colunas têm de ter um domínio – Ex. string, integer, real
  - SGBD oferecem domínios específicos Ex. char(n), int, smallint, number(n,m)
- Grau da relação = número de colunas
- Cardinalidade de uma relação = número de linhas

## Síntese de Comandos SQL

- SQL-DDL: Data Definition Language - operações sobre a estrutura das tabelas e gestão de restrições de integridade
  - CREATE TABLE
  - DROP TABLE
  - ALTER TABLE
- SQL-DML: Data Manipulation Language - operações sobre os dados das tabelas
  - INSERT INTO
  - DELETE FROM
  - UPDATE
  - SELECT

## Criação de Tabelas

- Dado um esquema de relação  
Students (sid: integer, name: string, login: string, age: integer, gpa: real)
- O comando CREATE TABLE cria uma nova tabela sem dados

```
CREATE TABLE Students (  
  sid INTEGER,  
  name CHAR(30),  
  login CHAR(30),  
  age INTEGER,  
  gpa REAL);
```

## Inserção em Tabelas

- O comando INSERT insere dados numa tabela
- Atributos inseridos devem ser especificados pois podem ser feitas inserções incompletas

```
INSERT INTO Students (sid,  
name, login, age, gpa)  
VALUES  
  
(53668, 'Smith', 'smith@ee',  
18, 3.2);
```

## Atualização de Tabelas

- O comando UPDATE atualiza dados existentes na tabela

```
UPDATE Students S  
SET S.age=S.age+1,  
S.gpa=S.gpa-1  
WHERE S.sid =53688
```

## Remoção em Tabelas

- O comando DELETE apaga dados numa tabela

```
DELETE  
FROM Students S  
WHERE S.name = 'Smith'
```

## Restrições de Integridade

- Condições especificadas sobre o esquema da BD
  - Restringe os dados que podem ser armazenados numa instância
  - Impede o armazenamento de informação incorreta ou incoerente
- Verificação automática pelo SGBD
- Integridade de domínio
  - Cada coluna de uma tabela tem um domínio de dados (ex. INTEGER)
  - Todos os valores dessa coluna têm de pertencer ao mesmo domínio
- Integridade de coluna
  - Refinamento da integridade de domínio
  - Permite limitar gama de valores admissíveis

## Restrições de Chave

- Chaves candidatas são colunas (atributos) com as seguintes propriedades
  - Unicidade: os seus valores identificam univocamente qualquer tuplo de uma instância (dois tuplos distintos não podem ter valores iguais para os atributos da chave)
  - Minimalidade: conjunto mínimo de atributos que identificam univocamente qq tuplo de uma instância, i.e, nenhum subconjunto de atributos da chave pode ser uma chave
- Podem existir várias chaves candidatas por relação
  - Existe sempre uma chave candidata
  - Chave primária é uma das chaves candidatas selecionada como a principal
  - Normalmente do tipo INTEGER

```
CREATE TABLE Empregado (  
  nid    INTEGER(4) PRIMARY KEY,  
  nif    INTEGER(9) UNIQUE NOT NULL, ...)
```

↖ chave candidata

## Integridade Referencial

- Chave estrangeira
  - Restrição de integridade que envolve duas tabelas
  - Denominada restrição de integridade referencial
  - Coluna(s) cujos valores provêm da chave primária de outra tabela
  - Se os dados de uma relação são alterados, as outras relações devem ser verificadas para manter os dados consistentes
- Propriedades:
  - Cada valor de studid que aparece na tabela Enrolled tem de aparecer na coluna da chave primária da tabela Students
  - Operações que podem originar violações
    - Inserir linhas em Enrolled
    - Remover linhas de Students •
  - A chave estrangeira pode referenciar a própria tabela

```
CREATE TABLE Enrolled(  
    studid INTEGER(10),  
    cid CHAR(20),  
    grade CHAR(1),  
    PRIMARY KEY (studid, cid),  
    FOREIGN KEY (studid)  
    REFERENCES Students (sid))
```

## Valor NULL

- NULL indica que para aquele campo o valor é desconhecido ou não aplicável
- NULL pode aparecer numa chave estrangeira sem violar a restrição de integridade referencial
  - Se a chave estrangeira for constituída por várias colunas, ou estão todas a NULL ou nenhuma
- NULL não pode aparecer na chave primária

## Restrições Gerais

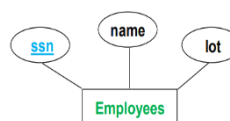
- Exemplo: as idades dos estudantes têm de ser maiores que 18
  - CHECK (age > 18)
- SGBD rejeita remoções/atualizações que violem as restrições:
  - Restrições de tabela: envolvem uma única tabela
  - Asserções: envolvem várias tabelas

## Passagem de EA para Relacional

- Modelo entidade-associação (EA)
  - Adequado para o desenho inicial, de alto nível, da base de dados
  - Representação gráfica para facilitar discussão de alternativas por equipas
  - Não entendido pelos sistemas de gestão de bases de dados (SGBD)
- Modelo relacional
  - Suportado pelos SGBDs relacionais, muito populares
  - Baixo nível, com comandos de texto, que dificultam discussão
  - Maior risco de perder a visão do todo, focando apenas nas partes
- Após discussão de alternativas e integração de diagramas EA
  - Esquema EA é traduzido num esquema relacional (ER) aproximado
  - Com tabelas e restrições de integridade escritas na linguagem SQL
  - Algumas restrições de integridade EA podem não ser concretizadas em SQL

## Entidades para ER

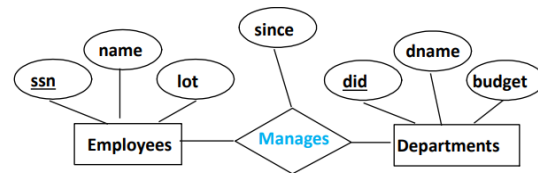
- Nome da tabela igual ao nome da entidade
- Colunas da tabela são os atributos da entidade
- Chave primária da tabela vem da chave primária da entidade



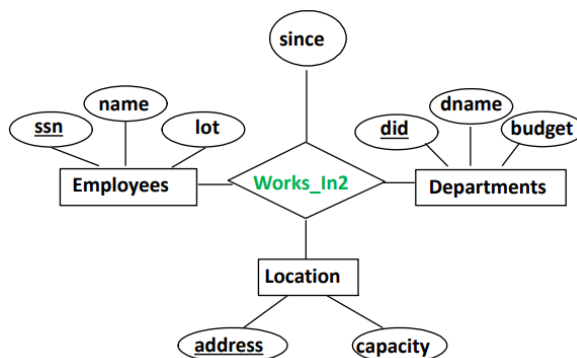
```
CREATE TABLE Employees (  
    ssn CHAR(11),  
    name CHAR(30),  
    lot INTEGER,  
    PRIMARY KEY (ssn));
```

## Associações para ER

- Caso 1: sem restrições de chave e participação
  - Nome da tabela igual ao nome da associação
  - Chave primária da tabela é composta pelas chaves primárias das entidades participantes
  - Chaves estrangeiras referenciam as entidades
- Exemplo com ternária

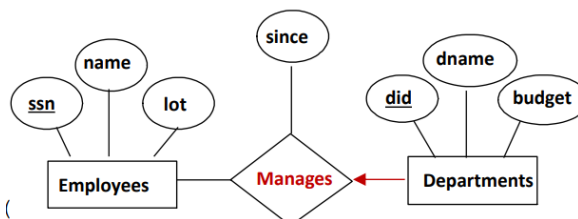


```
CREATE TABLE Manages (
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (did,ssn),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments);
```



```
CREATE TABLE Works_In2 (
  ssn CHAR(11),
  did INTEGER,
  address CHAR (20)
  since DATE,
  PRIMARY KEY (did, ssn, address),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments
  FOREIGN KEY (address) REFERENCES Location);
```

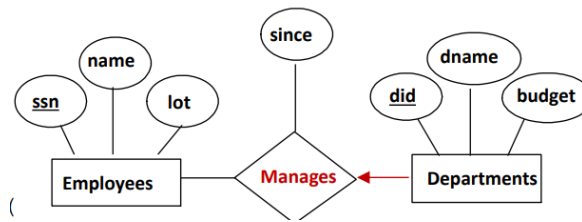
- Caso 2: com restrição de chave
  - Abordagem 1: criação de uma nova tabela
  - Vantagens
    - Atributos descritivos da associação na sua própria tabela
    - Restrição de participação parcial facilmente suportada: basta não inserir linhas na tabela
  - Desvantagens
    - Mais uma tabela no esquema relacional torna pesquisas mais complexas
    - Restrição de participação total custosa: necessárias asserções
    - Usar em casos de associações com muitos atributos descritivos



```
CREATE TABLE Manages (
  ssn CHAR(11) NOT NULL,
  did INTEGER,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments);
```

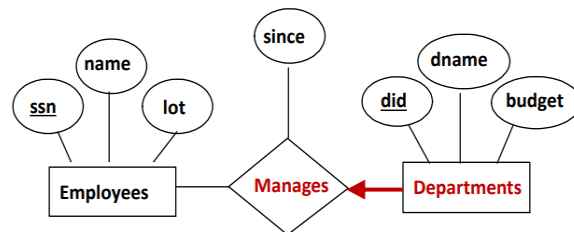
- Abordagem 2: Adição de chave estrangeira à tabela existente
- Vantagens
  - Menos uma tabela no esquema relacional permite pesquisas mais simples
  - Restrição de participação total facilmente suportada: basta usar NOT NULL
- Desvantagens

- Mistura atributos da associação e entidade na mesma tabela
- Restrição de participação parcial pode levar a muitos valores nulos nas linhas da tabela



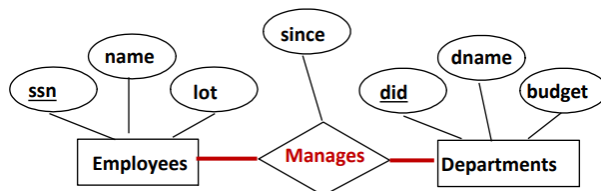
```
CREATE TABLE Dept_Mgr (
  ssn CHAR(11),
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees);
```

- Caso 3: com restrição de chave e participação
  - ON DELETE NO ACTION: ação por defeito, um empregado não pode ser removido se tiver um Dept\_Mgr a referenciá-lo.



```
CREATE TABLE Dept_Mgr (
  ssn CHAR(11) NOT NULL,
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees
  ON DELETE NO ACTION);
```

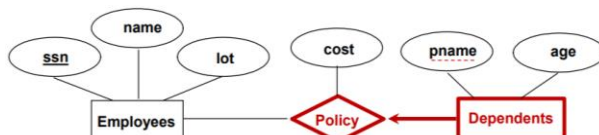
- Caso 4: com restrição de participação
  - Criação de nova tabela e asserção (RI) – “Cada departamento tem de ter pelo menos um empregado (e vice-versa)”



```
CREATE TABLE Manages (
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (did, ssn),
  FOREIGN KEY (ssn) REFERENCES Employees
  FOREIGN KEY (did) REFERENCES
  Departments);
```

## Entidades Fracas para ER

- Tabela com chave estrangeira para entidade forte
- Chave primária da entidade fraca composta por chave parcial e chave primária da entidade forte
- ON DELETE CASCADE: Remoção de linha na entidade forte despoleta a remoção das respectivas linhas na entidade fraca.



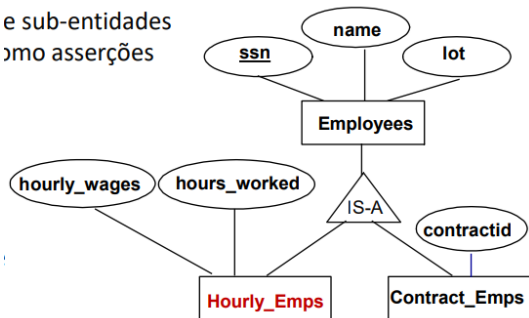
```
CREATE TABLE Dep_Policy (
  pname CHAR(20),
  age INTEGER,
  cost REAL,
  ssn CHAR(11),
  PRIMARY KEY (pname, ssn),
  FOREIGN KEY (ssn) REFERENCES Employees
  ON DELETE CASCADE);
```



## Generalizações para ER

- Abordagem 1 - Criação de tabelas para a super-entidade e sub-entidades
  - Restrições de cobertura e sobreposição como asserções
  - Nas tabelas das sub-entidades - Chave primária vem da super-entidade
  - Sempre aplicável
  - Necessário consultar duas tabelas para obter todos os dados de cada sub-entidade

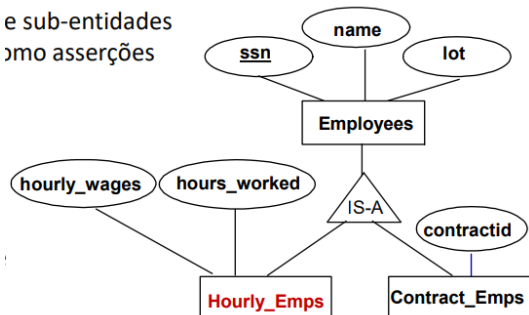
e sub-entidades  
como asserções



```
CREATE TABLE Hourly_Emps (
  hours_worked INTEGER,
  hourly_wages REAL,
  ssn CHAR(11),
  PRIMARY KEY (ssn),
  FOREIGN KEY (ssn) REFERENCES Employees
  ON DELETE CASCADE);
```

- Abordagem 2 - Criação de tabelas apenas para as sub-entidades
  - Restrições de cobertura e sobreposição como asserções
  - Apenas aplicável quando existe cobertura total
  - Mais eficiente para interrogações a sub-entidades específicas
  - Restrições de cobertura e sobreposição como asserções

e sub-entidades  
como asserções



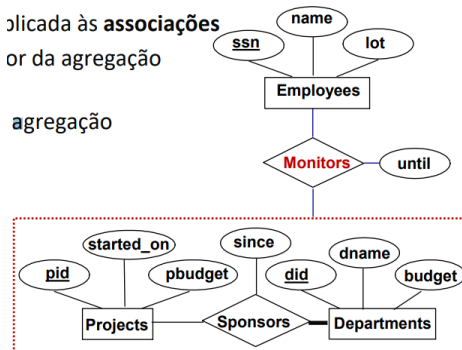
```
CREATE TABLE Hourly_Emps (
  ssn CHAR(11),
  name CHAR(30),
  lot INTEGER,
  hours_worked INTEGER,
  hourly_wages REAL,
  PRIMARY KEY (ssn));
```

```
CREATE TABLE Contract_Emps (
  ssn CHAR(11),
  name CHAR(30),
  lot INTEGER,
  contractid INTEGER,
  PRIMARY KEY (ssn));
```

## Agregações para ER

- Passagem semelhante à aplicada às associações
- Primeiro traduz-se o interior da agregação – Sponsors
- Depois a associação com a agregação – Monitors

aplicada às associações  
ou da agregação



```
CREATE TABLE Monitors (
  ssn CHAR(11),
  pid INTEGER,
  did INTEGER,
  until CHAR(11),
  PRIMARY KEY (ssn, pid, did)
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (pid, did) REFERENCES Sponsors);
```

## Alteração de Tabelas

- Alteração de tabelas com o comando ALTER TABLE
- Adição de coluna
  - Linhas que já existem ficam a NULL nessa coluna
  - Ex. ALTER TABLE Students  
ADD COLUMN maiden-name CHAR(10)
- Alteração de coluna
  - Novo domínio deverá abranger valores já existentes nessa coluna
  - Ex. ALTER TABLE Students  
MODIFY COLUMN maiden-name CHAR(20)
  - Atenção na alteração de colunas: se o novo domínio for menor do que os anteriores podem ser perdidos dados existentes na tabela
- Remoção de uma coluna
  - Ex. ALTER TABLE Students  
DROP COLUMN maiden-name
- Adição de restrição de integridade
  - Apenas se todos os dados já existentes cumprirem a nova regra
  - Ex. ALTER TABLE Enrolled  
ADD CONSTRAINT nn\_enrolled\_grade CHECK (grade IS NOT NULL)
- Remoção de restrição de integridade
  - Tem efeito permanente
  - Ex. ALTER TABLE Enrolled  
DROP CONSTRAINT nn\_enrolled\_grade

## Pesquisas de Dados

- Pesquisas de dados com o comando SELECT
  - Seleção de linhas e colunas de uma ou mais tabelas
  - Ex. SELECT \*  
FROM Students
  - Ex. (duas tabelas)  
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid = E.studid AND grade = 'A'

## Vistas sobre Dados - tabela virtual

- Cujas linhas não são explicitamente armazenadas
- Conteúdo determinado na criação da view por um comando SELECT
- Pode mostrar apenas algumas colunas e linhas da(s) tabela(s) de base
- Essencial para independência e privacidade de dados
- Pode abstrair alterações na tabela de base
- Cada tipo de utilizador pode ter acesso a vistas específicas
- Ex. CREATE VIEW Old\_students (name, login, age) AS  
SELECT name, login, age  
FROM students  
WHERE age >= 18
- Consulta da view executa a interrogação SQL associada
- SQL distingue entre vistas
  - Cujas linhas podem ser modificadas (updatable views)
  - Onde novas linhas podem ser inseridas (insertable views)
  - Tem de existir uma relação de um para um entre as linhas da vista e das linhas das respetivas tabelas

## Inserção de Dados nas Views

```
CREATE VIEW good_student (sid, gpa) AS
SELECT sid, gpa
FROM student
WHERE gpa >= 3 WITH CHECK OPTION

INSERT INTO good_student VALUES
(10000, 3),
(11000, 1.8);
```

Nota: CHECK OPTION verifica se os valores introduzidos na view respeitam as condições da sua criação.

## Remoção de Tabelas e Vistas

- Remoção de tabela e dos seus dados
  - As chaves estrangeiras para esta tabela têm de ser removidas antes
  - Ex. DROP TABLE Student
  - Ex. DROP TABLE Students RESTRICT – Apaga a tabela exceto se existirem views ou restrições de integridade referencial
  - Ex. DROP TABLE Students CASCADE – Apaga a tabela e todas as views e restrições de integridade referencial
  - Remoção de view – DROP VIEW Good\_students

## Forma Básica de uma Interrogação SQL

```
SELECT [DISTINCT] select-list
FROM from-list
[WHERE qualification]
```

select-list: lista de colunas a selecionar

from-list: lista de uma ou mais tabelas de onde provêm os dados

qualification: condições para definir as linhas a selecionar condição booleana que admite AND, OR, NOT, {<=, =, <>, >=, >}

DISTINCT: elimina linhas duplicadas

## Avaliação de uma Interrogação

1. Calcula o produto cartesiano de todas as tabelas no from-list (FROM)
  2. Elimina as linhas que falham a condição qualification (WHERE)
  3. Elimina as colunas que não aparecem na select-list (SELECT)
  4. Elimina linhas duplicadas se usar DISTINCT
- Ex. Marinheiros com um rating maior que 7  
SELECT S.sid, S.sname, S.rating, S.age  
FROM Sailors S  
WHERE S.rating > 7
  - Sinónimo S pode ser usado no contexto do SELECT em vez de Sailors (etiquetagem de tabela)
  - SELECT \* seria uma alternativa para mostrar todas as colunas
  - SELECT \* É aceitável em modo interativo
  - Em programação de aplicações com BD é preferível indicar explicitamente
  - Com duas ou mais tabelas é essencial usar a condição de junção  
Ex. SELECT S.sname  
FROM Sailors S, Reserves R  
WHERE S.sid = R.sid AND R.bid=103

## Expressões na Select-list e Qualification

- Select-list pode ter mais do que nomes de colunas de tabelas
  - Expressões aritméticas e chamadas a funções (ex. funções de agregação)
  - Cada expressão deve ter um nome fácil de interpretar
    - Ex. `SELECT Sailors.rating + 1 AS new_rating ...`
- Na Qualification as condições podem incluir
  - Expressões aritméticas e chamadas a funções
    - Ex. Nome dos marinheiros com o dobro do rating de outros marinheiros  
`SELECT S1.sname AS name1, S2.sname AS name2  
FROM Sailors S1, Sailors S2  
WHERE 2*S1.rating = S2.rating`

## Operador LIKE

- O operador LIKE suporta uma variante de expressões regulares
  - O caracter % representa zero ou mais caracteres arbitrários
  - O caracter \_ representa um caracter arbitrário
  - O espaço é importante no LIKE
  - Ex. Idade dos marinheiros cujo nome começa por um qualquer caracter, seguido de um A, depois de um B e depois um qualquer outro caracter  
`SELECT S.age  
FROM Sailors S  
WHERE S.name LIKE '_AB%'`

## Construtores de conjuntos

- União  
`SELECT ... UNION SELECT ...`
  - União das linhas dos dois conjuntos
- Interseção  
`SELECT ... INTERSECT SELECT ...`
  - Linhas comuns a ambos os conjuntos
- Diferença  
`SELECT ... EXCEPT SELECT ...`
  - Linhas de um conjunto às quais se retiraram as linhas de outro conjunto
- Por omissão são eliminadas as linhas duplicadas
- Para manter os duplicados, `UNION ALL`, `INTERSECT ALL`, `EXCEPT ALL`
- Solução sem union e com union

```
SELECT S.name  
FROM Sailors S, Reserves R, Boats B  
WHERE S.sid = R.sid AND R.bid = B.bid  
AND (B.color='red' OR B.color='green')
```

```
SELECT S.name  
FROM Sailors S, Reserves R, Boats B  
WHERE S.sid = R.sid AND R.bid = B.bid  
AND B.color='green'  
UNION  
SELECT S.name  
FROM Sailors S, Reserves R, Boats B  
WHERE S.sid = R.sid AND R.bid = B.bid  
AND B.color='red'
```

- Solução sem intersect e com intersect

```
SELECT S.name
FROM Sailors S, Reserves R1, Boats B1
     Reserves R2, Boats B2
WHERE S.sid = R1.sid AND R1.bid =
B1.bid AND S.sid = R2.sid AND R2.bid = B2.bid
AND (B1.color='red' AND B2.color='green')
```

```
SELECT S.name
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND B.color='red'
INTERSECT
SELECT S2.name
FROM Sailors S2, Reserves R2, Boats B2
WHERE S2.sid = R2.sid AND R2.bid =
B2.bid AND B2.color='green'
```

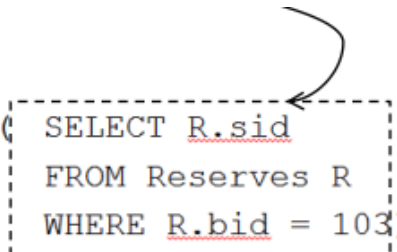
- Exemplo de except

```
SELECT R.sid
FROM Reserves R, Boats B
WHERE R.bid = B.bid AND B.color='red'
EXCEPT
SELECT R2.sid
FROM Reserves R2, Boats B2
WHERE R2.bid = B2.bid AND
B2.color='green'
```

## Sub-Interrogações

- Uma interrogação pode conter outras interrogações
  - Um (ou mais) SELECT dentro de um SELECT
  - Também denominadas nested queries
  - A interrogação que está embebida chama-se sub-interrogação

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN (SELECT R.sid
                   FROM Reserves R
                   WHERE R.bid = 103)
```



- Numa interrogação podem aparecer
  - Na cláusula WHERE (o mais frequente)
  - Na Cláusula FROM
  - Na cláusula HAVING
- Independentes (operadores IN e NOT IN)
- Correlacionadas (operadores EXISTS e NOT EXISTS)

## Interrogação com Operador IN

- Nomes dos marinheiros que reservaram o barco 103  
SELECT S.sname  
FROM Sailors S  
WHERE S.sid IN ( SELECT R.sid  
                  FROM Reserves R  
                  WHERE R.bid = 103)
  - A sub-interrogação devolve o conjunto dos identificadores dos marinheiros que reservaram o barco 103
  - A interrogação seleciona apenas os marinheiros que pertencem ao conjunto da sub-interrogação

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN ( SELECT R.sid
                  FROM Reserves R
                  WHERE R. bid IN (SELECT B.bid
                                   FROM Boats B
                                   WHERE B.color = 'red'))
```

## Interrogação com Operador NOT IN

- Nomes dos marinheiros que não reservaram barcos vermelhos

```
SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN( SELECT R.sid
                    FROM Reserves R
                    WHERE R. bid IN ( SELECT B.bid
                                       FROM Boats B
                                       WHERE B.color = 'red'))
```

## Exists

- Verifica a existência de valores no resultado da sub-interrogação
- A condição na sub-interrogação tem em conta o marinheiro atual na interrogação principal
- Nomes dos marinheiros que reservaram o barco 103
- A utilização de SELECT \* é recomendada nestas situações
- Outro caso – NOT EXISTS no WHERE: Marinheiros que não reservaram o barco 103

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS ( SELECT *
               FROM   Reserves R
               WHERE  R.bid = 103 AND
                     R.sid = S.sid )
```

## Comparação de Conjuntos com ANY

- Marinheiros cujo rating é maior que o rating de algum dos marinheiros chamados Horatio
  - Sub-interrogação devolve os ratings dos marinheiros Horatio
  - Interrogação seleciona os marinheiros cujo rating seja superior a algum em (1)
  - Se sub-interrogação devolve conjunto vazio então > ANY (empty) = false
  - SOME é sinónimo de ANY

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ANY
      ( SELECT S2.rating
        FROM Sailors S2
        WHERE S2.sname = 'Horatio' )
```

## Comparação de Conjuntos com ALL

- Marinheiros cujo rating é maior que o rating de todos os marinheiros chamados Horatio
  - Sub-interrogação devolve os ratings dos marinheiros Horatio
  - Interrogação seleciona os marinheiros cujo rating seja superior a todos em (1)
  - Se sub-interrogação devolve conjunto vazio então > ALL (empty) = true
  - Se não há Horatios então devolve todos os marinheiros

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ALL
      ( SELECT S2.rating
        FROM Sailors S2
        WHERE S2.sname = 'Horatio' )
```

## Exemplo de Escolha do Valor Máximo

- Marinheiros com o maior rating

```
SELECT S.sid
FROM Sailors S
WHERE S.rating >= ALL ( SELECT S2.rating FROM Sailors S2)
```
- Observações
  - WHERE ... IN .... equivalente a WHERE ... = ANY ...
  - WHERE ... NOT IN .... equivalente a WHERE ... <> ALL ...

## Operadores de Agregação

- Produzem sumários de dados tipicamente referentes a uma coluna da tabela
- Devolvem um valor único como resultado
- Operadores de agregação em SQL
  - COUNT ([DISTINCT] coluna) - Número de valores na coluna da tabela
  - SUM ([DISTINCT] coluna) - Soma dos valores na coluna A
  - AVG ([DISTINCT] coluna) - Média dos valores na coluna A – MAX (coluna) e MIN(coluna) Máximo e mínimo valor na coluna
- A utilização do \* no COUNT é recomendada quando estão a ser contadas linhas e não colunas específicas

## Interrogações com AVG

- Média de idades dos marinheiros  

```
SELECT AVG(S.age)
FROM Sailors S
```
- Média de idades dos marinheiros com rating de 10  

```
SELECT AVG(S.age)
FROM Sailors S
WHERE S.rating = 10
```

## Interrogações com MAX

- Idade do marinheiro mais velho  

```
SELECT MAX(S.age)
FROM Sailors S
```
- ```
SELECT S.sname, MAX(S.age)
FROM Sailors S
```

Não é um comando válido O operador de agregação MAX agrega num só valor os valores das outras colunas não estão acessíveis
- Solução:  

```
SELECT S.sname, S.age
FROM Sailors S
WHERE S.age = ( SELECT MAX(S2.age) FROM Sailors S2 )
```

## Sub-Interrogações com Operadores de Agregação

- Nomes dos marinheiros que são mais velhos que o marinheiro mais velho com rating de 10  

```
SELECT S.sname
FROM Sailors S
WHERE S.age > (SELECT MAX(S2.age) FROM Sailors S2 WHERE S2.rating = 10)
```
- Outra alternativa, com operador ALL  

```
SELECT S.sname
FROM Sailors S
WHERE S.age > ALL( SELECT S2.age FROM Sailors S2 WHERE S2.rating = 10)
```

## Agrupamento e Filtragem

- Sintaxe do comando  

```
SELECT [DISTINCT] select-list
FROM from-list
[WHERE qualification]
GROUP BY grouping-list
[HAVING group-qualification]
```
- GROUP BY permite criar grupos de linhas
  - Cada grupo de linhas tem o mesmo valor nas colunas do grouping-list
  - Ex. GROUP BY (age) cria tantos grupos quantas as idades existentes
- HAVING elimina os grupos que não satisfazem a condição da group-qualification
  - Tem de incluir colunas da grouping-list ou operadores de agregação
  - Ex. HAVING (age > 15) inclui grupos cujos marinheiros têm idade > 15
- Cada coluna na select-list só pode ter um valor único por grupo
  - Podem ser colunas do grouping-list
  - Podem ser operadores de agregação



## Interrogações com GROUP BY e HAVING

Idade do marinheiro mais novo com **mais de 18 anos** para cada rating, cuja média de idades dos marinheiros (com mais de 18 anos) seja superior à **média de idade de todos os marinheiros**?

```
SELECT S.rating, MIN(S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING AVG(S.age) > ( SELECT AVG(S2.age)
                      FROM Sailors S2)
```

## Operador de divisão

- Utilidade
  - Permite expressar interrogações de um determinado tipo
  - p.e., "Nome dos marinheiros que reservaram todos os barcos"
- Exemplo

|   |     |     |    |     |        |     |
|---|-----|-----|----|-----|--------|-----|
| A | sno | pno | B1 | pno | A / B1 | sno |
|   | s1  | p1  |    | p2  |        | s1  |
|   | s1  | p2  | B2 | pno |        | s2  |
|   | s1  | p3  |    | p2  |        | s3  |
|   | s1  | p4  |    | p4  |        | s4  |
|   | s2  | p1  | B3 | pno | A / B2 | sno |
|   | s2  | p2  |    | p1  |        | s1  |
|   | s3  | p2  |    | p2  | A / B3 | sno |
|   | s4  | p2  |    | p4  |        | s1  |
|   | s4  | p4  |    |     |        |     |

© 2021 - Docentes SI - DI/FCUL

Fonte: António Ferreira, Guião SIBD, 2016

- 2 Soluções:

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE R.sid = S.sid
GROUP BY S.sname
HAVING COUNT(DISTINCT R.bid)=( SELECT COUNT(*)
                              FROM Boats B)
```

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (
  SELECT B.bid
  FROM Boats B
  WHERE NOT EXISTS (
    SELECT R.bid
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid ))
```

## Ordenação de Resultados de Interrogações

- Sintaxe do comando  
SELECT [DISTINCT] select-list  
FROM from-list  
[WHERE qualification]  
[GROUP BY grouping-list]  
[HAVING group-qualification]  
ORDER BY order-list
- ORDER BY permite ordenar os resultados de uma interrogação
  - Sem ORDER BY a ordem das linhas no resultado é arbitrária
  - As colunas na order-list devem constar na select-list
  - Ordenação ascendente (ASC) ou descendente (DESC)

## Valores Nulos

- Usados quando se desconhece o valor de uma coluna ou quando não aplicável
  - Em SQL são representados por NULL
  - Ex. desconhece-se o telemóvel de um cliente, mas pode vir a saber-se
  - Ex. nome de solteira só é relevante para mulheres casadas (maiden name)
- Aplicabilidade
  - Podem ser usados em colunas com qualquer domínio de dados
  - Não podem ser usados em chaves candidatas, chave primária
  - Não podem ser usados em colunas NOT NULL
- Operações com Valores Nulos
  - Operações de comparação (<=, =, <>, >=, >) e aritméticas (+, -, \*, /)
  - Basta um dos argumentos ser NULL, para resultado ser unknown
  - Operações conjunção e disjunção (AND, OR)
    - NULL AND FALSE = FALSE, cc. resultado é sempre unknown
    - NULL OR TRUE = TRUE, cc. resultado é sempre unknown
  - Verificação de valores nulos (IS NULL)
    - NULL IS NULL = TRUE (ex. s.age IS NULL)
    - NULL IS NOT NULL = FALSE (ex. s.age IS NOT NULL)
- Impacto nos Comandos SQL
  - A condição na cláusula WHERE
    - Só aparecem no resultado as linhas que verificam a condição ou seja, para a qual a condição seja TRUE
    - Elimina as linhas cujo resultado seja FALSE ou unknown
- Operadores de agregação
  - COUNT(\*) , na contagem inclui os valores NULL
  - Restantes operadores (com ou sem DISTINCT) ignoram NULL  
COUNT(coluna), SUM(coluna), AVG(coluna), MAX/MIN(coluna)
  - Se todos os valores na coluna forem nulos, resultado é NULL  
Exceto COUNT(coluna) que devolve 0

## Junções Naturais (natural join)

```
SELECT *  
FROM Sailors S, Reserves R  
WHERE S.sid = R.sid
```

```
SELECT *  
FROM Sailors S INNER JOIN Reserves R  
ON (S.sid = R.sid)
```

## **Junções Exteriores (outer join)**

- As junções exteriores – Devolvem as linhas que satisfazem a condição de junção e as linhas numa das tabelas sem correspondência com as da outra tabela
- FULL OUTER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN
- Variantes - Considerando a junção de T1 com T2, por esta ordem
  - Junção exterior esquerda: inclui todos os valores de T1
  - Junção exterior direita: inclui todos os valores de T2
  - Junção exterior completa: inclui todos os valores de T1 e T2

## **Violação de Integridade Referencial**

- Remoção (DELETE) e atualização (UPDATE) de linhas em tabelas referenciadas
  - Pode gerar problemas nas chaves estrangeiras das tabelas referenciadoras
  - Exemplo: Aluno deixa de existir ou muda de número
- Opções para preservação da integridade referencial  
FOREIGN KEY .... REFERENCES... ON DELETE (ou UPDATE)
  - NO ACTION - opção por omissão Linhas apenas são removidas/atualizadas se não referenciadas na chave estrangeira
  - CASCADE - Linhas na tabela referenciadora também são apagadas/atualizadas –
  - SET NULL - Os valores correspondentes da chave estrangeira são colocados a NULL

## **Verificação de Restrições de Integridade**

- Modo imediato
  - O SGBD verifica as RI após a execução de cada comando SQL
  - Comando SQL: SET CONSTRAINTS ... IMMEDIATE
- Modo diferido
  - O SGBD verifica as RI no final da execução de uma transação  
Comando SQL: SET CONSTRAINT ... DEFERRED
  - Útil para operações interdependentes que temporariamente criam incoerência Exemplo: Departamento tem obrigatoriamente um chefe. Empregado pertence obrigatoriamente a um departamento

## **Regras de Integridade - Resumo**

- Integridade de Domínio
  - Cada atributo de uma relação tem um domínio
  - O valor desse atributo para todos os tuplos terá de pertencer SEMPRE a esse domínio ou ser NULL
- Integridade da Chave
  - Dois tuplos distintos de uma relação não podem ter um conjunto de valores iguais nos atributos da chave
  - Nenhum subconjunto de atributos de uma chave é uma chave candidata
- Integridade de Entidade
  - Nenhum atributo componente de uma chave primária poderá em algum momento ter valor NULL
- Integridade Referencial

Numa relação, qualquer ocorrência de uma chave estrangeira deverá obrigatoriamente:

  - Existir como ocorrência da chave primária da relação à qual se refere.
  - Ou ter todos os atributos NULL.
- Integridade Aplicacional (adicional ou semântica)
  - Qualquer outra regra a que as ocorrências de uma determinada base de dados deverão obedecer e que não é abrangida pelos tipos atrás mencionados.

## Definição de Novos Domínios

- Definição de novos domínios
  - Para manter coerência nas colunas
  - Ex. novo domínio com nºs inteiros entre 1 e 10  

```
CREATE DOMAIN ratingval INTEGER  
DEFAULT 1  
CHECK ( VALUE >= 1 AND VALUE <= 10 )
```
- Aplicação do novo domínio em coluna de tabela
  - Ex. 

```
CREATE TABLE Sailors ( ...,rating ratingval, ...)
```
  - Se INSERT em Sailors omitir o valor de rating, este é preenchido com 1
- Valores de rating podem ser comparados com os de colunas do tipo INTEGER
  - Limitação conceptual, pois os domínios são diferentes

## Definição de Tipos

- Comando CREATE TYPE define um novo tipo de dados abstrato
- Necessita de métodos próprios para suportar comparações, adições, ... mesmo que baseado em domínios simples, como INTEGER
- Para evitar comparações entre tipos diferentes  
Exemplo: 

```
CREATE TYPE ratingtype AS INTEGER
```

## Domain vs Type

- Colunas ratingtype não podem ser comparadas (ou operadas) com colunas do tipo INTEGER
- Colunas ratingval podem ser comparadas (ou operadas) com colunas do tipo INTEGER

## Restrições Complexas numa Tabela

- Definição de tabelas pode incluir cláusulas CHECK  

```
CREATE TABLE Sailors ( sid INTEGER, ...,rating INTEGER,  
CHECK (rating >= 1 AND rating <= 10 ))
```
- Definição de tabelas pode incluir restrições mais complexas
  - Com a consulta a outras tabelas
  - Ex.: reservas de barcos, exceto para barcos com nome Interlake  

```
CREATE TABLE Reserves (sid INTEGER, bid INTEGER, day DATE,  
PRIMARY KEY (sid, bid),  
FOREIGN KEY (sid) REFERENCES Sailors (sid),  
FOREIGN KEY (bid) REFERENCES Boats (bid),  
CONSTRAINT noInterlakeRes  
CHECK ('Interlake' <> ( SELECT B.bname  
FROM Boats B  
WHERE B.bid = Reserves.bid )))
```
  - Condição verificada para cada INSERT ou UPDATE na tabela Reserves

## Asserções

- Restrições que não estão associadas a uma qualquer tabela
- Definidas ao mesmo nível das tabelas no esquema de dados
- Apropriadas para restrições que abrangem múltiplas tabelas
- Ex.

```
CREATE ASSERTION smallClub  
CHECK ((SELECT COUNT (S.sid) FROM Sailors S)  
+(SELECT COUNT (B. bid) FROM Boats B)<100)
```

## Triggers

- Procedimento que é automaticamente despoletado quando se realizam escritas específicas
  - Evento de escrita ativa condição que permite, ou não, execução de ação
- Evento – Tipo de escrita na base de dados que faz ativar o trigger
  - Tipos de escrita: qualquer combinação de INSERT, UPDATE, e DELETE
  - Escritas podem ser numa tabela inteira ou em colunas específicas
  - Opções de ativação do trigger
    - Antes ou depois da escrita se concretizar
    - Uma só vez para um bloco inteiro de escritas ou para cada linha escrita
- Condição (opcional)
  - Uma interrogação ou um teste verificado aquando da ativação do trigger
- Ação
  - Código do procedimento executado quando o trigger é ativado e a condição anterior satisfeita

## MySQL: exemplo de trigger

```
CREATE TRIGGER upd_check
BEFORE UPDATE ON account
FOR EACH ROW
BEGIN
    IF NEW.amount < 0 THEN
        SET NEW.amount = 0;
    ELSEIF NEW.amount > 100 THEN
        SET NEW.amount = 100;
    END IF;
END;
```

## Bases de Dados Ativas e Triggers

- Base de dados ativa
  - Base de dados com triggers associados
- Usos típicos de triggers
  - Restrições de integridade complexas
  - Autorizações de acesso e auditoria de escritas em tabelas Ex. que utilizadores escreveram em certa tabela e a que horas
  - Réplicas síncronas de tabelas
- Definição de triggers – responsável e problemas
  - Tipicamente definidos (ou autorizados) pelo DBA = DataBase Administrator
  - Razão: consequências do uso de triggers podem ser difíceis de entender
  - Vários triggers podem ser ativados em simultâneo, por ordem arbitrária
  - Ação de um trigger pode ativar outros triggers (triggers recursivos)
  - Um uso criterioso de restrições de integridade pode frequentemente substituir/evitar uso de triggers

## Checks vs. Triggers

- CHECKS
  - Declarativos
  - Mais fáceis de entender
  - Mais eficientes pois podem ser otimizados pelos SGBDs
  - Restrições de integridade verificadas em permanência
- Triggers
  - Procedimentais (requer saber programar)
  - Desempenho depende da qualidade do programador
  - Podem ser usados para outros fins (além de manter integridade)
    - Autorizações de acesso, auditoria, estatísticas, replicação de dados...
  - Restrições de integridade verificadas para escritas específicas em tabelas
  - Necessário cuidado para cobrir todos os cenários possíveis de escrita de dados

## Normalização

- Normalização permite melhorar a qualidade do esquema através da eliminação da redundância dos dados e prevenção de anomalias
- Normalização é uma abordagem que envolve a decomposição sucessiva de relações até se obter um conjunto de relações sem redundâncias e que permitam inserções, atualizações e remoções sem incoerências

## Problemas da Redundância dos Dados

A redundância introduz problemas (anomalias) de coerência e manutenção

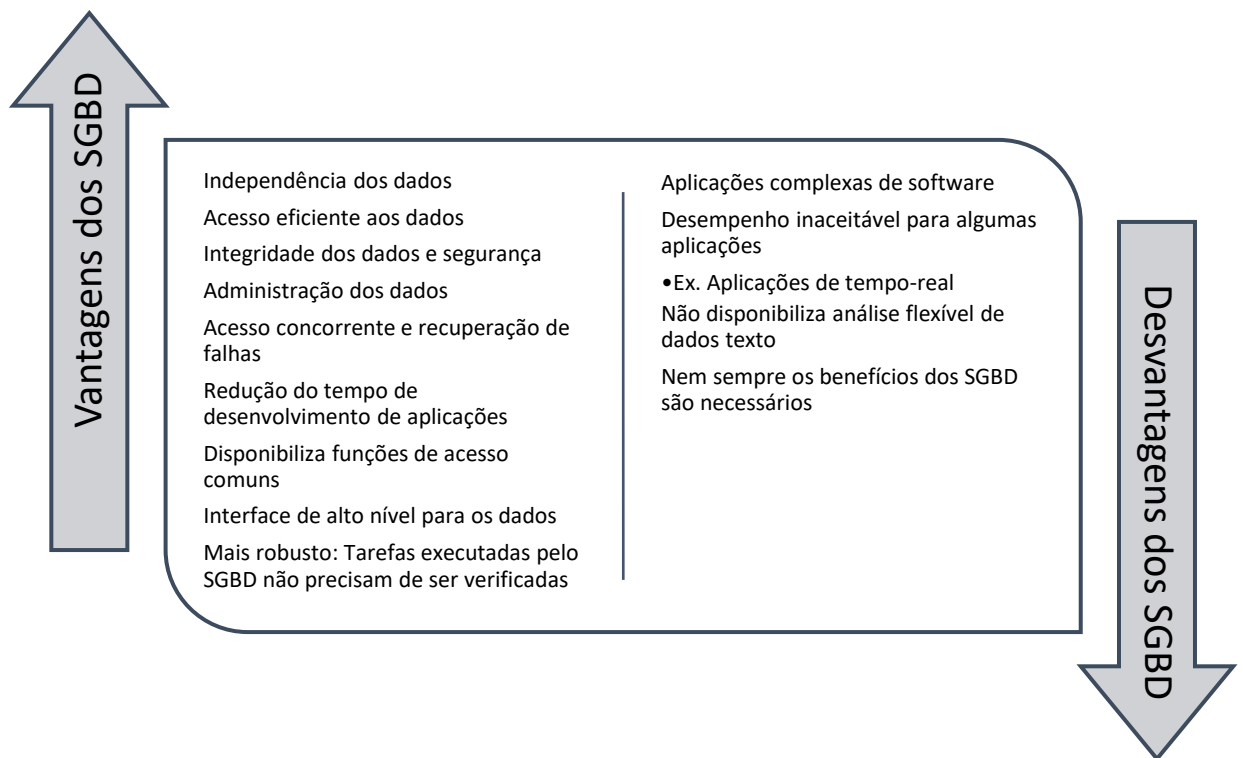
- Anomalia de inserção - informação que é independente não pode ser inserida de forma separada na Base de Dados
- Anomalia de atualização - a modificação de informação num conjunto de ocorrências implica a criação de inconsistências ou a necessidade de alterar informação noutras instâncias da Base de Dados que são independentes das primeiras
- Anomalia de remoção - a remoção de informação acarreta a perda de outra informação independente contida na Base de Dados

## Principais Formas Normais

- Primeira forma normal: 1FN
  - Colunas da relação guardam apenas um valor por linha
  - Tipicamente verificado em BD relacionais
- Segunda forma normal: 2FN = 1FN +
  - Colunas não pertencentes às chaves candidatas da relação dependem da totalidade das colunas de cada chave
  - Trivial, se chaves da relação tiverem apenas uma coluna
- Terceira forma normal: 3FN = 2FN +
  - Colunas não pertencentes às chaves candidatas dependem apenas das chaves candidatas

## Razões de Uso de SGBDs

- Suportam conceitos úteis para a gestão de BDs
  - Linguagem para criar, alterar, e consultar/analisar dados
  - Restrições para manter a integridade dos dados (ex. idade > 0)
  - Utilizadores e privilégios de acesso aos dados
  - Transações: programas que transformam o estado da BD
- Oferecem mecanismos automáticos de gestão de BDs
  - Aplicação de restrições de integridade aquando de alterações nos dados
  - Controlo de acesso à BD por vários utilizadores em simultâneo
  - Recuperação de faltas (ex. falta de energia elétrica)
  - Otimização do acesso aos dados, para respostas rápidas
- Suportam grandes quantidades de dados e de transações
  - Mecanismos desenhados para permitir expansão de capacidade



### Bases de Dados noSQL

- NoSQL é uma abordagem recente para criar bases de dados
  - o termo NoSQL é usado para referir bases de dados não relacionais

- são BD não tabulares e armazenam os dados de maneira diferente das tabelas relacionais
- as BD NoSQL surgem em vários tipos com base no seu modelo de dados p. e.: documento, chave-valor (key-value), graph,...
- desenhadas para serem escaláveis para grandes quantidades de dados, grandes quantidades de utilizadores
- Existem vários produtos NoSQL – exs: MongoDB, Neo4j, Apache Cassandra, Google Cloud BigTable

### **Conceção de BDs**

- Como se pode descrever um caso real em termos de dados a guardar num SGBD?
- Que fatores devem ser considerados na decisão de como organizar os dados armazenados?
- Como desenvolver aplicações que recorrem a um SGBD?

### **Análise dos Dados**

- Como o utilizador pode obter resposta às suas questões através de interrogações aos dados no SGBD?

### **Concorrência e Robustez**

- Como é que um SGBD permite o acesso concorrente aos dados?
- Como é que este protege os dados num caso de uma falha do sistema? (ex. falta eletricidade)

### **Eficiência e Escalabilidade**

- Como é que um SGBD armazena grandes colecções de dados e executa interrogações nesses dados de uma forma eficiente? (ex. índices)
- Como lida com grande quantidade de utilizadores e acessos?

### **SQL em Aplicações**

- Os comandos SQL podem ser chamados através de uma aplicação informática

### **Concorrência e Consistência**

- Requisitos de um sistema transacional
  - Gestão de múltiplas transações em simultâneo
  - Gestão das regras de integridade
- Concorrência
  - Múltiplos utilizadores e respetivos pedidos em simultâneo
- Consistência (Integridade)
  - A BD está num estado consistente quando cumpre as regras de integridade
- Um SGBD é um sistema transacional

### **Execução Concorrente de Transações**

- Um protocolo de locking é um conjunto de regras que devem ser seguidas por cada transação (garantido pelo SGBD)
  - Apesar das operações das transações serem executadas intercaladamente
  - O resultado é o mesmo de executar as transações numa determinada ordem
- Dois tipos de locks:
  - Partilhados por múltiplas transações (leitura)
  - Exclusivos a uma transação (escrita e leitura)
- Alterações feitas por transações incompletas devem ser anuladas
- O SGBD mantém um log de todas as operações de escrita:



- Para anular transações incompletas
  - Ou refazer transações completas depois de uma falha
- Checkpoints (forçam escrita em disco) para minimizar o tempo de recuperação

### **Desenvolvimento e Uso de SGBDs**

- Intervenientes
  - Empresas que desenvolvem os SGBDs (DB implementors)
  - Programadores (DB application programmers)
  - Utilizadores de aplicações (end users)
  - Administradores de bases de dados (DB Administrator - DBAs)

### **Tarefas Críticas do Administrador BD**

- Concepção do modelo lógico e físico
  - Interage com os utilizadores para decidir que relações (tabelas) armazenar e como
- Fiabilidade
  - Faz backups periódicos
  - mantém logs da atividade do sistema
- Afinação – Adequa o desempenho às alterações de requisitos
- Segurança e autorização
  - Atribui diferentes permissões de acesso
  - 4 tipos de acesso aos recursos (table/view): CRUD Create, Read, Update, Delete
  - Utilização diferenciada para diferentes roles
 

Um utilizador pode ter vários roles p.e., app\_programador, app\_read, app\_write

Cada role está associado a privilégios

```
CREATE ROLE 'app_programador' , 'app_read' , 'app_write'
```

```
GRANT ALL ON appdb.* TO 'app_programador';
```

```
GRANT SELECT ON appdb.* TO 'app_read';
```

```
GRANT INSERT, UPDATE, DELETE ON appdb.* TO 'app_write';
```