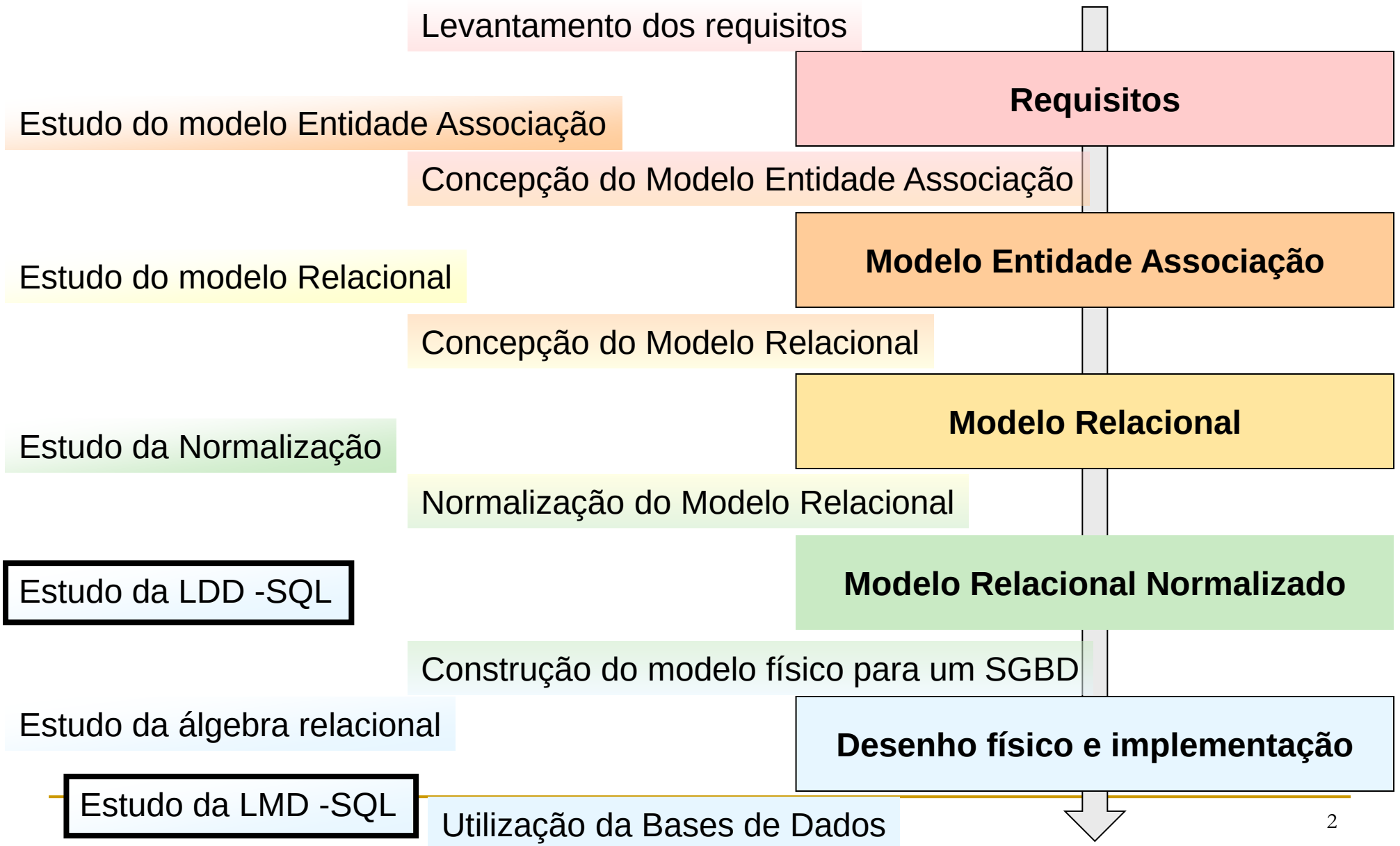
06 – Linguagem SQL

Baseado nos slides do professor Paulo Trigo
Todas as alterações são da responsabilidade do professor António Teófilo

Etapas do processo



O que é / como usar o SQL ?

- SQL (Structured Query Language) é uma linguagem normalizada, utilizada nos SGBD para realizar acções de:
  - Interrogação
  - Manipulação de Dados
  - Definição de Dados
  - Controlo de Transações
  - Autorizações e Segurança
- Na maioria dos SGBD Relacionais, o SQL pode ser utilizado:
  - Interactivamente
  - Embutido em Linguagens de Programação
    - existem diversas extensões em que os comandos SQL podem ser embutidos em linguagem de programação tradicionais (C, C++, etc).

Componentes da linguagem SQL

- LDD - Linguagem de Definição de Dados
  - do inglês DDL - “Data Definition Language”
  - Permite:
    - construir o Esquema da base de dados
- LMD - Linguagem de Manipulação de Dados
  - do inglês DML - “Data Manipulation Language”
  - Permite:
    - aceder à informação armazenada na base de dados
    - inserir nova informação na base de dados
    - eliminar informação na base de dados
    - alterar informação na base de dados

Características da linguagem SQL

- Unifica a LDD e a LMD numa única linguagem
- Inclui aspectos relacionados com a administração da base de dados
- É um standard da ISO e da ANSI
- É uma Linguagem não-Procedimental
  - Especifica-se O QUÊ e não COMO
- Existe uma clara abstracção perante a estrutura física dos dados
  - não é necessário especificar caminhos de acesso
  - não é necessário elaborar algoritmos de pesquisa física
- As operações efectuam-se sobre:
  - conjuntos de Dados (Tabelas)
  - não é necessário (nem possível) manipular os Dados Linha-a-Linha.

Perspectiva Histórica

- 1970: “Codd” define o Modelo Relacional
- 1974: IBM desenvolve o SYSTEM/R com a linguagem SEQUEL
  - (Structured English Query Language) cujo nome depois evoluiu para SQL
- Lançamento de SGBDs comerciais
  - 1979: Primeiro SGDB comercial (ORACLE)
  - 1981: SGBD INGRES
  - 1983: IBM anuncia o DB2
- Normas SQL
  - 1986, 1987: Norma SQL-87 (ANSI X3.135-1986 e ISO 9075:1987)
  - 1989: Norma SQL-89 (ANSI X3.135.1-1989)
  - 1992: Norma SQL-92 ou SQL2 (ISO/IEC 9075:1992)
  - 1999: Norma SQL:1999 ou SQL3
    - Added regular expression matching, recursive queries, triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features.
  - 2003: Norma SQL:2003
    - Introduced XML-related features, window functions, standardized sequences, and columns with auto-generated values
  - 2006: Norma SQL:2006
    - ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it provides facilities that permit applications to integrate into their SQL code the use of XQuery, to concurrently access ordinary SQL-data and XML documents.

SQL - Alguns comandos principais

- Interrogação
  - **SELECT**. Usado para interrogar a BD.
- Manipulação de Dados
  - **INSERT**, **UPDATE**, **DELETE**. Usados para inserir novos registos em tabelas, alterar ou eliminar registos já existentes.
- Definição de Dados
  - **CREATE**, **ALTER**, **DROP**. Usados para criar, alterar ou eliminar qualquer estrutura de dados (tabelas, vistas e índices).
- Controlo de Transações
  - **COMMIT**, **SAVEPOINT**, **ROLLBACK**. Usados para controlar explicitamente as transacções.
- Autorizações e Segurança
  - **GRANT**, **REVOKE**. Usados para atribuir ou retirar direitos de acesso.

Definição de dados

criar, alterar, apagar tabelas

DDL – Linguagem de Definição de Dados

Definição do Esquema da Base de Dados

- Comandos SQL pertencentes à Linguagem de Definição de Dados (LDD):
  - CREATE, para criar uma tabela
  - ALTER, para alterar uma tabela existente
  - DROP, para eliminar uma tabela a existente

- Uma tabela pode ser criada em qualquer momento de uma sessão SQL
- A estrutura de uma tabela pode ser alterada em qualquer momento de uma sessão SQL
- Toda a informação eliminada não pode ser recuperada

- Uma tabela não tem qualquer dimensão predeterminada

Tipos de Dados

- O Standard SQL2:
  - BIT(*n*), BIT VARYING(*n*)
  - CHARACTER(*n*), CHARACTER VARYING(*n*)
  - DATE, TIME, TIMESTAMP
  - DECIMAL, NUMERIC
  - FLOAT, REAL, DOUBLE PRECISION
  - INTEGER, SMALLINT
- No caso do SQL Server:
  - Binários: BINARY[(n)], VARBINARY[(n)]
  - Caracter: CHAR[(n)], VARCHAR[(n)]
  - Data e Hora: DATETIME, SMALLDATETIME (ao milissegundo, ao minuto)
  - Numérico exacto: DECIMAL[(p[,s])], NUMERIC[(p[,s])] (numeric = decimal)
  - Numérico aproximado: FLOAT[(n)], REAL (n bits of mantissa (1-53), default 53, real = float(24)
  - Inteiro: BIGINT, INT, SMALLINT, TINYINT (respectivamente 8, 4, 2 e1 bytes)
  - Monetário: MONEY, SMALLMONEY (respectivamente 8 e 4 bytes)

Tipos de Dados

- No caso do MySQL:
  - Binários: BIT[(n)]
  - Caracter: CHAR[(n)], VARCHAR[(n)], ENUM, TEXT
  - Data e Hora: DATE, DATETIME, TIME, YEAR, TIMESTAMP
  - Numérico exacto: DECIMAL[(p[,s])], NUMERIC [(p[,s])],
  - Numérico aproximado: FLOAT[(n, d)], DOUBLE[(m, d)]
  - Inteiro: BIGINT, INT, MEDIUMINT, SMALLINT, TINYINT

MEDIUMINT - 3 bytes

Tipos de Dados (cont.)

- No caso do SQL Server, existe ainda:
  - Texto e Imagem: TEXT, IMAGE
  - Especiais: BIT, tipos de dados definidos pelo utilizador
- Algumas características de tipos de dados (no caso de SQL Server):
  - Para os tipos de dados VAR...(n) apenas é alocado o espaço necessário para guardar o dado, até ao máximo definido (n), que pode ter o valor de *max* (2^{31}-1 bytes/caracteres).
  - Para o correspondente tipo sem o prefixo VAR, é alocado todo o espaço definido (n), independentemente da dimensão efectiva do dado.
    - Pelo facto de ocupar um espaço fixo, o acesso ao tipo VAR... pode ser mais rápido do que ao correspondente tipo sem o prefixo VAR
    - É de utilizar o tipo VAR quando se prevê, nessa coluna, existência de valores nulos ou grandes variações na dimensão dos dados
  - BINARY(n), n grupos de 2 dígitos hexadecimais, para guardar dados binários
  - DATETIME, data e hora (8bytes), com resolução ao milissegundo (3.33 ms) *
  - SMALLDATETIME, data e hora (4bytes), com resolução ao minuto *

* desde 1/1/1900

Tipos de Dados (cont. 1)

- Algumas características de tipos de dados (no caso de SQL Server):
  - DECIMAL e NUMERIC são idênticos e existem ambos por razões de compatibilidade.
  - DECIMAL[(p[,s])] - o parâmetro “p” (*precision*) indica o número máximo total de dígitos; o parâmetro “s” (*scale*) indica o número máximo de dígitos à direita do ponto decimal.
    - No SQL Server *precision* pode ir de 1 a 38, com valor por omissão de 18. *Scale* por ir até 0 a *precision*, sendo o valor por omissão de 0.
    - Espaço ocupado: varia em função de *precision*/p, sendo 5 se p $\in$ [1 – 9], 9 se p $\in$ [10-19], 13 se p $\in$ [20-28] e 17 se p $\in$ [29-38].
  - FLOAT[(n)] – *n* indica o número de bits para guardar a mantissa.
    - No SQL server, o *n* pode variar entre 1 e 53, sendo 53 o seu valor por omissão, mas o seu valor é tido como 24 se for inferior ou igual a 24, e 53 se for superior a 24. O espaço ocupado é de 4 bytes para n = 24 e 8 bytes para n = 53, tendo respectivamente uma precisão de 7 e 15 dígitos.
    - O tipo REAL, corresponde a FLOAT(24)

Criação de uma tabela

- Comando CREATE

```
CREATE TABLE nome_tabela (
  nome_coluna  tipo [restrição_coluna], ...
  [restrição_tabela, …]
  [AS SELECT comando]
)
```

- Exemplo criar tabela e colunas

```
CREATE TABLE ITEMFACTURA (
  numFactura int,
  numLinha int,
   descricaoProd varchar(30),
  quantidade int NOT NULL DEFAULT 1
 )
```

NOT NULL, indica que a coluna não pode receber o valor NULL
DEFAULT 1, indica que o valor por omissão é 1

Criação de uma tabela - restrições

- restrição_coluna (aplica-se apenas a uma coluna)
- restrição_tabela (aplica-se a uma ou mais colunas)
- Definição de uma restrição:

CONSTRAINT nome_restrição
[PRIMARY KEY | **UNIQUE]** (coluna, …) |
[**FOREIGN KEY** (coluna, …) {REFERENCES tabela [(coluna, …)]
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}] |
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}] } |
[**CHECK** (condição)]

- ON DELETE/UPDATE: NOACTION, CASCADE, SET DEFAULT, SET NULL

Remoção de um tuplo que tenha tuplos a referencia-lo, por intermédio de uma chave estrangeiras, resulta em:
ON DELETE NOACTION→ a acção de DELETE não é realizada
ON DELETE CASCADE→ na remoção desses tuplos
ON DELETE SET DEFAULT→ na colocação das referências com o valor por omissão
ON DELETE SET NULL → na colocar dessas referências a NULL

Criação de uma tabela – exemplos

- Definição de uma tabela com uma chave primária, uma chave candidata (alternativa) e restrição de valor NULL por CHECK:

```
CREATE TABLE ALUNO (
   numAluno int CONSTRAINT pk_ALUNO PRIMARY KEY,
   numBI decimal(8,0) NOT NULL CONSTRAINT ak1_ALUNO UNIQUE,
   nome varchar(100),
   CONSTRAINT ck1_ALUNO CHECK ( nome IS NOT NULL )
)
```

- Definição de uma tabela com chave primária composta:

```
CREATE TABLE FACTURA (
    numFactura int,
    numLinha int,
    produto varchar(50) not null,
    quantidade int not null,
    CONSTRAINT pk_FACTURA PRIMARY KEY ( numFactura, numLinha )
)
```

Criação de uma tabela – exemplos

- Definição de uma tabela com chave estrangeira composta:

```
CREATE TABLE ENTREGA (
  numEntrega int,
  dataEntrega datetime NOT NULL,
  numFactura int NOT NULL,
  numLinha int NOT NULL,
  CONSTRAINT pk_ENTREGA PRIMARY KEY ( numEntrega ),
  CONSTRAINT fk1_ENTREGA FOREIGN KEY ( numFactura, numLinha )
    REFERENCES FACTURA ( numFactura, numLinha )
    ON DELETE CASCADE
)
```

Considere-se que cada item de factura pode ser entregue separadamente. Não será o caso normal.

- ON DELETE CASCADE indica que,
  - a remoção de uma linha da tabela FACTURA ,
  - implica a remoção das linhas da tabela ENTREGA que lhe estiverem associadas

Criação de uma tabela – exemplos

- Definição de uma tabela com uma regra de verificação (CHECK):

```
CREATE TABLE ENCOMENDA (
  numEnc int CONSTRAINT pk_ENCOMENDA PRIMARY KEY,
  dataEnc datetime NOT NULL,
  codCliente int NOT NULL CONSTRAINT fk1_ENCOMENDA
    FOREIGN KEY (codCliente) REFERENCES CLIENTE (codCliente),
  dataEntrega datetime,
  CONSTRAINT ck1_ENCOMENDA CHECK (dataEntrega IS NOT NULL AND
   dataEntrega > dataEnc ))
```

- Definição de uma tabela com valores seleccionados de outra tabela:

```
CREATE TABLE EMPREGADO_AUX (
  (codigoEmpregado, nomeEmpregado, codigoCategoria)
  AS SELECT codEmp, nome, codCat FROM EMPREGADO
)
```

1. Cria a tabela **Empregado_aux** com as colunas codigoEmpregado, nomeEmpregado e codigoCategoria, que irão ter domínios idênticos respectivamente às colunas codEmp, nome e codCat da tabela **Empregado**.
2. A tabela **Empregado_aux** é preenchida com o resultado do select, ou seja com todos os empregados.

No SQL Server: **select** codDep, nomeDep **into** departamento_aux **from** departamento **where** ...;

Alterações a uma tabela

- Comando ALTER

  ALTER TABLE nome_tabela
  [ADD novas colunas | novas restrições_coluna]
  [MODIFY definição das colunas]
  [DROP coluna | restrição_coluna]

- Algumas alterações possíveis:
  - acrescentar colunas, eventualmente acompanhadas de restrições
  - acrescentar restrições de integridade à tabela
  - modificar o tipo de determinada coluna
  - remover uma restrição da tabela

  - É uma boa política criar as tabelas e depois adicionar as chaves estrangeiras com ALTER, pois fica independente da ordem pela qual as tabelas são criadas. Exemplo:

    Alter table INSCRICAO add constraint fk_aluno foreign key (numeroAluno) references ALUNO (numero)

Alterações a uma tabela (cont. 1)

- Algumas limitações às alterações:
  - Não se pode modificar uma coluna com valores NULL para NOT NULL.
  - Só se pode adicionar uma coluna NOT NULL a uma tabela que não contenha nenhuma linha.
    - solução: adicionar a coluna como NULL, preenche-la completamente e depois mudar para NOT NULL
  - Pode-se decrementar o tamanho de uma coluna e o tipo de dados, caso essa coluna tenha valor NULL em todas as linhas.
- Exemplos de alterações:

```
ALTER TABLE EMPREGADO
  ADD  comissao int NOT NULL;


ALTER TABLE DEPARTAMENTO
  MODIFY  nome varchar( 50 ) NOT NULL;
```

Alterações a uma tabela (cont. 2)

- Exemplos de alterações:
  - Eliminar uma coluna:

```
ALTER TABLE EMPREGADO
   DROP comissao
```

  - Eliminar uma restrição de integridade:

```
ALTER TABLE ENCOMENDA
   DROP CONSTRAINT ck1_ENCOMENDA
```

  - Acrescentar uma restrição de integridade (chave estrangeira):

```
ALTER TABLE ENCOMENDA
   ADD CONSTRAINT fk1_ENCOMENDA FOREIGN KEY codCliente
          REFERENCES CLIENTE ( codCliente )
```

Remoção de uma tabela

- Comando DROP

```
DROP TABLE nome_tabela
```

- Exemplo de remoção da tabela EMPREGADO

```
DROP TABLE EMPREGADO
```

- Algumas características da acção de remoção de uma tabela:
  - a remoção de uma tabela causa a perda de todos os dados nela existentes, assim como de todos os índices associado
  - uma tabela só pode ser removida por quem a criou ou pelo administrador da Base de Dados

Manipulação de dados
inserir, alterar, remover linhas

Manipulação de Dados

- Comandos SQL pertencentes à Linguagem de Manipulação de Dados (LMD):
  - **INSERT** - insere linhas

```
INSERT INTO <nome da tabela> [(coluna1, coluna2, ...)]
VALUES (valor1, valor2, ...)
[comando SELECT]
```

  - **UPDATE** - actualiza colunas

```
UPDATE <nome da tabela>
SET coluna=valor | expressão, coluna=valor | expressão, ...
[WHERE <condição>]
```

  - **DELETE** - remove linhas

```
DELETE FROM <nome da tabela>
[WHERE <condição>]
```

Inserção de linhas numa tabela

- Inserção directa de dados
- Pretende-se registar a existência do departamento de Marketing, localizado em Lisboa e com código 4

```
INSERT INTO DEPARTAMENTO ( codDep, nome, localizacao )
 VALUES( 4, 'Marketing', 'Lisboa' )
```

- Características do comando INSERT (aplicado deste modo):
  - apenas se pode inserir uma linha de cada vez *
  - pode-se omitir a lista de nomes de colunas, desde que se respeite a ordem das colunas definidas na tabela e não se omita nenhum valor
  - pode-se inserir uma linha apenas com algumas das colunas da tabela, ficando os campos omitidos com valores nulos
  - É boa política indicar os nomes das colunas, pois fica-se com independência face a alterações na definição da tabela

Multirow inserts, since SQL-92 (DB2, MySQL, H2, PostgreSQL)

```
INSERT INTO table (column1, [column2, ... ])
VALUES (value1a, [value1b, ...]) [,  (value2a, [value2b, ...]) ] *
```

Exemplo:

```
INSERT INTO phone_book VALUES ('John Doe', '555-1212'), ('Peter Doe', '555-2323');
```

Insert MultiRow no SQL Server

Uma forma de se conseguir um insert multi-linha, quando tal não existe, é fazer a junção dos dados através do comando UNION

```
insert into aeronave
   (codAeronave, fabricanteAeronave, modeloAeronave, autonomia)
   select 'AIRBUS-A580', 'AIRBUS', 'A580', '5000' union all
   select 'AIRBUS-A480', 'AIRBUS', 'A480', '4000' union all
   select 'AIRBUS-A380', 'AIRBUS', 'A380', '3000' union all
   select 'BOIENG-747' , 'BOIENG', '747` , '5500';
```

Inserção de linhas numa tabela (cont.)

- Inserção de dados provenientes de uma tabela
- Pretende-se:
  - copiar para a tabela com informação histórica dos empregados, todos os empregados dos departamentos de Lisboa.
  - o Esquema de Relação que irá conter a informação histórica dos empregados é: HISTORICO_EMPREGADO(codEmp, nome)

```
INSERT INTO HISTORICO_EMPREGADO( codEmp, nome )
SELECT codEmp, E.nome
  FROM EMPREGADO AS E INNER JOIN DEPARTAMENTO AS D
    ON (E.codDep = D.codDep )
  WHERE localizacao = 'Lisboa'
```

- Características do comando INSERT (aplicado deste modo):
  - o número de colunas na lista da cláusula SELECT tem que ser igual ao número de colunas referidas no comando INSERT e os domínios de colunas correspondentes têm que ser compatíveis

Actualização de linhas numa tabela

- Pretende-se registar o facto do empregado com código 123 ter mudado de departamento e de chefe. O seu novo departamento tem código 444 e o seu novo chefe tem código 654

```
UPDATE EMPREGADO
   SET codDep = 444, codEmpChefe = 654
   WHERE codEmp = 123
```

- Características do comando UPDATE:
  - se a cláusula WHERE for omitida, todas as linhas da tabela são actualizadas
  - os valores a actualizar podem ser o resultado de expressões ou interrogações à Base de Dados

Remoção de linhas numa tabela

- Pretende-se remover os empregados do departamento 222

```
DELETE FROM EMPREGADO
  WHERE codDep = 222
```

- Pretende-se remover toda a informação relativa ao histórico dos empregados

```
DELETE FROM HISTORICO_EMPREGADO
```

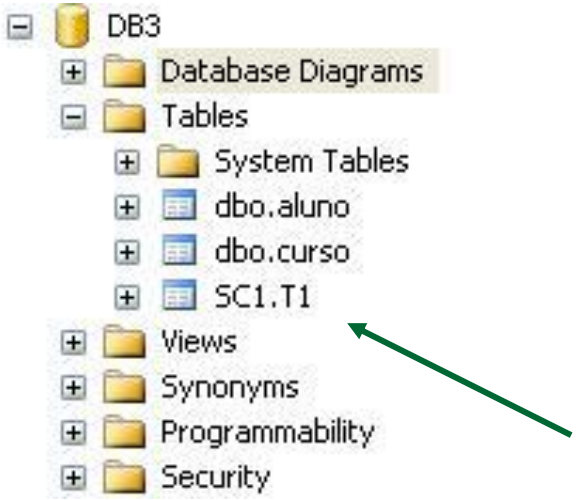
- Características do comando DELETE:
  - se a cláusula WHERE for omitida, todas as linhas da tabela são removidas

Mais notas acerca do SQL Server

- Aconselha-se uma vista a:
  - SQL Server 2005 Books Online (September 2007)
    - **Transact-SQL Reference (Transact-SQL)**
    - http://msdn.microsoft.com/en-us/library/ms189826(SQL.90).aspx
- Tópicos abordados:
  - Schemas
  - Funções
  - Constraint Unique
  - UniqueIdentifier
  - Computed columns

Schema – um espaço de nomes

```
use DB3;

create schema SC1;

create table SC1.T1 (
   nome varchar(120) not null,
   localidade varchar(40) not null
);

insert into SC1.T1 values ('Joaquim Lopez', 'Madrid');

select * from SC1.T1;

delete from SC1.T1;

drop table SC1.T1;

drop schema SC1;
```



O **schema** por omissão é: **dbo**

Um **schema** permite ter vários espaços de nomes para as tabelas

Funções (Functions)

- SQL Server 2005 Books Online (September 2007)
  - **Functions (Transact-SQL)**
  - http://msdn.microsoft.com/en-us/library/ms174318(SQL.90).aspx
  - Consultar o link para mais informações, esta lista serve apenas para alertar para a existência de funções

- Day, Month, Year, getdate
- Lower, upper, substring
- Rand, cos, tan, square, pi
- current_user, isdate, isnull
- case, top (não é uma função, mas é muito útil)

Constraint Unique e UniqueIdentifier

- Constraint Unique
  - Criar um *index* com valores únicos. Um *índex* permite acelerar as pesquisas que utilizem as colunas existentes no *índex*.
  - Pode-se aplicar UNIQUE na definição de uma coluna, ou aplicar como uma restrição (*constraint*) de uma tabela.

- UniqueIdentifier – tipo identificador único
  - É um tipo de identificador global e único: 16-byte GUID com a forma textual de: '*xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*'

```
CREATE TABLE MyUniqueTable (
   UniqueColumn UNIQUEIDENTIFIER DEFAULT NEWID(),
   Characters VARCHAR(10)
)
GO
INSERT INTO MyUniqueTable(Characters) VALUES ('abc')
INSERT INTO MyUniqueTable VALUES (NEWID(), 'def')
GO
```

Colunas auto-incrementáveis (Identity)

- Identity
  - permite declarar uma coluna com valores auto-incrementáveis,
  - podendo-se definir o valor inicial (valor da esquerda) e o passo do incremento (valor da direita)

```
CREATE TABLE new_employees (
   id int IDENTITY(1,1),
   firstname varchar (20),
   lastname varchar(30)
);

INSERT new_employees (firstname, lastname)
   VALUES ('Karin', 'Josephs');              -- ficará com id = 1
INSERT new_employees (firstname, lastname)
   VALUES ('Pirkko', 'Koskitalo');           -- ficará com id = 2
```

Colunas computadas (computed columns)

- Colunas computadas
  - São colunas virtuais, que não são fisicamente existentes na tabela, mas que existem como resultado da aplicação e uma expressão aplicada às restantes colunas da mesma tabela
  - Uma coluna computada pode ficar fisicamente registada caso se assinale como PERSISTED.
  - Exemplo:

```
CREATE TABLE FACTURA (
  numFactura int,
  numLinha int,
  produto varchar(50) not null,
  quantidade int not null,
  custoUnitario decimal(18,2) not null,
  custo AS custoUnitario * quantidade,
  CONSTRAINT pk_FACTURA PRIMARY KEY ( numFactura, numLinha )
)
```

Vistas (views)

Vistas (*Views*) da Base de Dados

- Usando a terminologia SQL, uma Vista (normalmente referida como *View*) consiste numa <u>única tabela</u> construída a partir de:
  - outras tabelas ou
  - outras Vistas anteriormente definidas.
- Características de uma Vista (*View*):
  - Uma Vista (*View*) não tem dados próprios.
  - Os dados de uma Vista (*View*) são manipulados na(s) tabela(s) base que servem de suporte a essa Vista.
  - Uma Vista (*View*) é armazenada como um comando SELECT.
- Uma Vista (*View*) é uma tabela virtual, isto é, não tem existência física embora apareça ao utilizador como se o tivesse, pelo que:
  - origina algumas limitações às operações de actualização (*update*);
  - não origina limitações às operações de interrogação (*select*).

Tipos de Vistas (*Views*) e sua utilidade

- Tipos de Vistas (*Views*)
  - Vistas simples:
    - construídas com base numa única tabela;
    - não contêm funções nem grupos de dados.
  - Vistas complexas:
    - construídas com base em várias tabela;
    - contêm funções ou grupos de dados.
- Utilidade das Vistas (*Views*):
  - restringir o acesso à Base de Dados mostrando apenas parte dos dados;
  - simplificar a consulta dos dados, substituindo consultas elaboradas envolvendo várias tabelas, por fáceis consultas sobre a Vista;
  - permitir que os mesmos dados sejam visualizados de diferentes maneiras por diferentes utilizadores;
  - reduzir a possibilidade de incoerências (opção WITH CHECK OPTION - alteração de dados através da Vista de acordo com a sua definição).

Criação de uma Vista – *View*

- Comando CREATE VIEW

  CREATE VIEW nome_vista

  [(nome_coluna_1, nome_coluna_2, ...)]

  AS SELECT comando [WITH CHECK OPTION]

- "nome_coluna_*i*":
  - nome da coluna usado na vista. Se não for especificado é assumido o mesmo nome das colunas definidas na cláusula SELECT.
- "AS SELECT comando" - consiste na directiva SELECT que define a vista, podendo usar mais do que uma tabela e outras vistas. Esta directiva tem algumas limitações:
  - não pode incluir as cláusulas ORDER BY
  - não pode incluir a instrução INTO
  - não pode referenciar uma tabela temporária.

Criação de uma Vista - *View* (Cont.)

- Considere-se o seguinte Esquema Relacional:
  - DEPARTAMENTO (nome, localizacao)
  - EMPREGADO (cod, nome, salario, nomeDep)
- Pretende-se criar uma vista que permita saber:
  - Para cada departamento quantos empregados existem e qual o montante total de salários.

```
CREATE VIEW INFORMACAO_DEPARTAMENTO
    (nomeDepartamento, numEmpregados, totalSalarios)
    AS SELECT D.nome, COUNT(*), SUM(salario)
    FROM DEPARTAMENTO AS D INNER JOIN EMPREGADO AS E
       ON ( D.nome = E.nomeDep )
    GROUP BY D.nome
```

- Esta vista é complexa: várias tabelas, funções e agregação de dados

Tratamento das Vistas pelo SGBD

- O comando CREATE VIEW não origina a execução do comando SELECT a ele associado.
- O comando CREATE VIEW apenas origina o armazenamento da definição da vista (directiva SELECT) no dicionário de dados.
- Ao aceder aos dados através de uma vista, o sistema:
  - extrai do dicionário de dados, a definição da vista;
  - verifica as permissões de acesso à vista;
  - converte a operação sobre a vista numa operação equivalente na tabela ou tabelas que servem de base à vista.

Interrogação a uma Vista - *View*

- Pretende-se, utilizando a vista INFORMACAO_DEPARTAMENTO, saber:
  - Para o departamento de Informática, quantos empregados existem e qual o montante total de salários.

```
SELECT numEmpregados, totalSalarios
   FROM INFORMACAO_DEPARTAMENTO
   WHERE nomeDepartamento = 'Informática'
```

- A vista não é concretizada no momento em que é criada, mas sempre que é especificada uma interrogação sobre essa vista, mantendo-se assim sempre actualizada.
- As alterações das tabelas originais reflectem-se nas diversas vistas dessas tabelas.

Remoção de uma Vista - *View*

- Comando DROP

```
DROP VIEW nome_vista
```

- Exemplo de remoção da vista INFORMACAO_DEPARTAMENTO

```
DROP VIEW INFORMACAO_DEPARTAMENTO
```

- Algumas características da acção de remoção de uma vista:
  - a remoção de uma vista não tem qualquer influência nos dados das tabelas que lhe serviam de base.
  - se existirem outras vistas que dependam da vista removida, essas vistas deixam de ser válidas.
  - uma vista só pode ser removida por um utilizador com permissões para efectuar essa operação ou pelo administrador da Base de Dados.

Actualização de dados através de uma Vista

- Se a vista for definida sobre:
  - uma única tabela e
  - sem funções de agregação de dados,
  - é uma operação simples que se traduz na actualização da tabela que lhe serve de base.
- Se a vista for definida sobre:
  - múltiplas tabelas e
  - com funções de agregação de dados,
  - é uma operação complicada e que pode ser ambígua.
- De uma forma geral, não são actualizáveis as vistas:
  - definidas sobre múltiplas tabelas utilizando junções (*join*);
  - que utilizam agrupamento de dados e funções de agregação.

Actualização de dados através de uma Vista (cont. 1)

- O comando de DELETE não é permitido se a vista incluir:
  - condições de junção (*join*);
  - funções de agrupamento;
  - o comando DISTINCT;
  - sub-interrogações correlacionadas.
- O comando de UPDATE não é permitido se a vista incluir:
  - qualquer das limitações do comando DELETE;
  - colunas definidas por expressões (tal como, salarioAno = 14 * salario).
- O comando de INSERT não é permitido se a vista incluir:
  - qualquer das limitações do comando UPDATE;
  - colunas com possibilidade de ter valores NOT NULL que não tenham valores de omissão nas tabelas base e que não estejam incluídas na vista através da qual se pretende inserir novas colunas.

Actualização de dados através de uma Vista (cont. 2)

- Os comandos de INSERT e UPDATE são permitidos em vistas contendo várias tabelas base se:
  - o comando afectar apenas uma das tabelas que serve de base à vista;
  - no entanto, não é permitido num mesmo comando alterar mais do que uma tabela.
- As vistas permitem executar verificações de integridade sobre os dados modificados através das vistas.
- A opção WITH CHECK OPTION indica que os comandos de INSERT e UPDATE executados através da vista só podem afectar registos seleccionáveis pela vista (definidos no WHERE).

Verificação de Integridade em Vistas

- Pretende-se criar uma vista que permita saber:
  - Quais os empregados que têm um salário inferior a 250000.
  - Também se pretende que qualquer acção de alteração sobre essa vista apenas afecte os empregados cujo salário seja inferior a 250000.

```
CREATE VIEW VISTA_EMPREGADO
  AS SELECT cod, nome
  FROM EMPREGADO
  WHERE salario < 250000
  WITH CHECK OPTION
```

- Devido ao Check Option as instruções de INSERT e UPDATE sobre a vista têm que verificar as condições definidas na cláusula WHERE.
  - a operação será rejeitada no caso dos dados não verificarem essas condições.

05 – Álgebra Relacional

Continuação: slides da álgebra relacional

Interrogação de dados
Comando Select

LMD - Linguagem de Manipulação de Dados

Select …

from …

where …

Operações Relacionais

- Operadores Unários
  - Restrição
  - Projecção
- Operadores Binários
  - União
  - Intersecção
  - Diferença
  - Produto Cartesiano
  - Junção
  - Divisão
- Operação Relacional
  - consiste na aplicação de um Operador Relacional a um conjunto de uma ou mais relações e tem como resultado uma nova relação.

Interrogação à Base de Dados

- Começando pela interrogação da Base de Dados, a sintaxe geral de uma questão SQL é a seguinte:

  SELECT [DISTINCT] <colunas> | *

  FROM <tabelas>

  [WHERE <condição>]

- Onde
  - <colunas> especifica a lista de atributos cujos valores interessa conhecer
  - <tabelas> especifica quais as tabelas envolvidas no processamento da interrogação
  - <condição> traduz a expressão lógica que define a condição a verificar
  - DISTINCT é aplicado a todas as colunas especificadas na cláusula SELECT e elimina as repetições existentes
  - O símbolo ***** é utilizado quando se pretendem seleccionar todos os atributos da tabela especificada na cláusula FROM

Projecção

EMPREGADO

codEmp	nomeEmp	dataAdmissao	codCat	codDept	codEmpChefe
1	António Abreu	01-Jan-99	1	1	1
2	Bernardo Bento	01-Jan-99	1	2	1
3	Carlos Castro	01-Jan-99	3	3	1
4	Manuel Matos	7-Feb-90	3	2	2

- Qual o código e nome de cada empregado ?
  - $\pi_{\text{codEmp, nomeEmp}}$ (EMPREGADO)

```
SELECT codEmp, nomeEmp
    FROM EMPREGADO
```

codEmp	nomeEmp
1	António Abreu
2	Bernardo Bento
3	Carlos Castro
4	Manuel Matos

Projecção (select distinct)

EMPREGADO

codEmp	nomeEmp	dataAdmissao	codCat	codDept	codEmpChefe
1	António Abreu	01-Jan-99	1	1	1
2	Bernardo Bento	01-Jan-99	1	2	1
3	Carlos Castro	01-Jan-99	3	3	1
4	Manuel Matos	7-Feb-90	3	2	2

- Quais as datas de admissão e código de categoria dos empregados?
  - π dataAdmissao, codCat (EMPREGADO)

```
SELECT dataAdmissao, codCat
    FROM EMPREGADO
```

dataAdmissao	codCat
01-Jan-99	1
01-Jan-99	1
01-Jan-99	3
7-Feb-90	3

- DISTINCT - garante que não existem tuplos duplicados

```
SELECT DISTINCT dataAdmissao, codCat
    FROM EMPREGADO
```

dataAdmissao	codCat
01-Jan-99	1
01-Jan-99	3
7-Feb-90	3

Selecção ou Restrição

CATEGORIA

codCat	designacao	salarioBase
1	CategoriaA	300
2	CategoriaB	250
3	CategoriaC	160

- Quais as categorias cujo salário base é superior a 250 ?
  - $\sigma_{\text{salarioBase} > 250}$ (CATEGORIA)

Operadores:
=, <, <=, >, >=, <>
OR, AND, NOT

```
SELECT *
    FROM CATEGORIA
    WHERE  salarioBase > 250
```

codCat	designacao	salarioBase
1	CategoriaA	300

- Pretende-se, não só informação das categorias designadas por 'CategoriaA', mas também das que têm salário base inferior a 200
  - $\sigma_{\text{designacao} = \text{'CategoriaA'} \vee \text{salarioBase} < 200}$ (CATEGORIA)

codCat	designacao	salarioBase
1	CategoriaA	300
3	CategoriaC	160

```
SELECT *
    FROM CATEGORIA
    WHERE designacao = 'CategoriaA' OR salarioBase < 200
```

Composição de operações de Restrição e Projecção

EMPREGADO

codEmp	nomeEmp	dataAdmissao	codCat	codDept	codEmpChefe
1	António Abreu	01-Jan-99	1	1	1
2	Bernardo Bento	01-Jan-99	1	2	1
3	Carlos Castro	01-Jan-99	3	3	1
4	Manuel Matos	7-Feb-90	3	2	2

- Qual o nome e data de admissão dos empregados da categoria com código 3, admitidos antes do dia 1 de Janeiro de 1998 ?
  - $\pi_{\text{nomeEmp, dataAdmissao}} (\sigma_{\text{codCat} = 3 \wedge \text{dataAdmissao} < \text{'01/01/1998'}} (\text{EMPREGADO}))$

```
SELECT DISTINCT nomeEmp, dataAdmissao
    FROM EMPREGADO
    WHERE codCat = 3 AND dataAdmissao < '01/01/1998'
```

nomeEmp	dataAdmissao
Manuel Matos	7-Feb-90

- Quais os nomes do empregados que são chefes deles próprios?
  - $\pi_{\text{nomeEmp}} (\sigma_{\text{codEmp} = \text{codEmpChefe}} (\text{EMPREGADO}))$

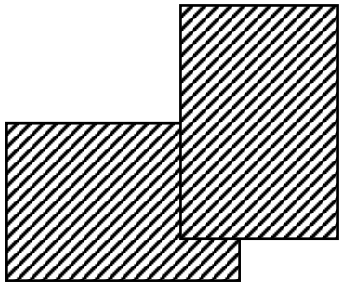
```
SELECT DISTINCT nomeEmp
    FROM EMPREGADO
    WHERE codEmp = codEmpChefe
```

nomeEmp
António Abreu

União

- Considere-se o seguinte Esquema Relacional
  - CLIENTE(<u>nome</u>, morada, telefone)
  - FORNECEDOR(<u>nome</u>, morada, telefone)
- Qual o conjunto que tem todos os clientes e todos os fornecedores ?
  - CLIENTE ∪ FORNECEDOR

```
SELECT *
    FROM CLIENTE
UNION
SELECT *
    FROM FORNECEDOR
```



- UNION - garante que não existem tuplos duplicados
- UNION ALL - inclui todos os tuplos com eventuais duplicados

União (cont.)

- A fim de fazer os convites para a festa de Natal em Lisboa, pretende-se a lista com todos os nome e telefones de todos os empregados e somente dos fornecedores de Lisboa, existentes nas tabelas EMPREGADO e FORNECEDOR

```
SELECT nome, telefone
    FROM EMPREGADO
UNION ALL
SELECT nome, telefone
    FROM FORNECEDOR
        WHERE localidade = 'Lisboa'
```

EMPREGADO

codEmp	nome	telefone
1	António Abreu	212555666
2	Bernardo Bento	212777888
3	Carlos Castro	212555444
4	Manuel Matos	212333111

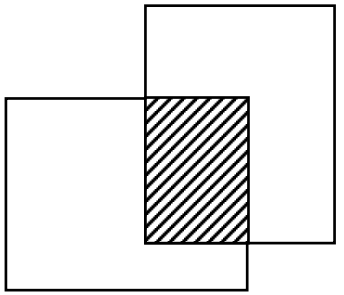
Fornecedor

nomeForn	telefone	localidade
Continentix	222111333	Lisboa
Jumbix	222222111	Lisboa
Feira Novix	222333444	Porto
Pingo Docix	222555111	Porto

nome	telefone
Continentix	222111333
Jumbix	222222111
António Abreu	212555666
Bernardo Bento	212777888
Carlos Castro	212555444
Manuel Matos	212333111

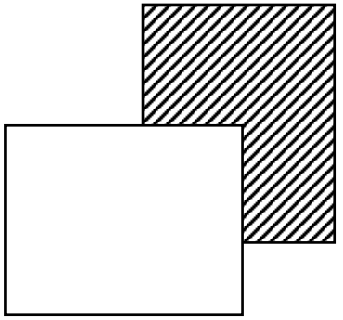
Intersecção e Diferença

- Quais os clientes que são também fornecedores ?
  - CLIENTE ∩ FORNECEDOR



```
SELECT *
    FROM CLIENTE
INTERSECT
SELECT *
    FROM FORNECEDOR
```

- Quais os clientes que não são fornecedores ?
  - CLIENTE - FORNECEDOR



```
SELECT *
    FROM CLIENTE
EXCEPT (ou MINUS em versões anteriores à SQL.92)
SELECT *
    FROM FORNECEDOR
```

União, Intersecção e Diferença

- Estas operações só podem ser aplicadas se os seus operandos (Relações) forem “Compatíveis em União”
- Duas Relações R e S são Compatíveis em União se:
  - tiverem o mesmo Grau (número de Atributos)
  - a cada Atributo em R corresponder outro em S com Domínio para o qual exista uma conversão implícita. A conversão típica é,
    - cadeias de caracteres convertidas na que tiver maior dimensão
    - tipos numéricos convertidos no que tiver maior precisão
- Exemplo em que as Relação <u>não são</u> Compatíveis em União
  - CLIENTE $\cup$ (π_{nome} (FORNECEDOR))

```
SELECT * FROM CLIENTE
UNION
SELECT nome FROM FORNECEDOR
```

Produto Cartesiano

FORNECE

codFornecedor	codMaterial
1	001
2	001
2	002
3	002

MATERIAL

numeroMaterial	nome
001	Parafuso
002	Roda Dentada

- Pretende-se uma lista onde cada código de fornecedor e cada código de material, apareça combinado com cada um dos materiais
  - FORNECE × MATERIAL

SELECT *

FROM FORNECE, MATERIAL

codFornecedor	codMaterial	numeroMaterial	nome
1	001	001	Parafuso
1	001	002	Roda Dentada
2	001	001	Parafuso
2	001	002	Roda Dentada
2	002	001	Parafuso
2	002	002	Roda Dentada
3	002	001	Parafuso
3	002	002	Roda Dentada

- Outra solução será,

SELECT *

FROM FORNECE CROSS JOIN MATERIAL

Produto Cartesiano e Restrição

- O Produto Cartesiano, por si só, tem uma utilidade limitada pois os registos são associados sem qualquer critério
- Pode ter mais interesse saber por exemplo,
  - Qual o material fornecido por cada fornecedor ?
  - $\sigma_{\text{codMaterial = numeroMaterial}}$ (FORNECE $\times$ MATERIAL)

```
SELECT *
    FROM FORNECE, MATERIAL
    WHERE codMaterial = numeroMaterial
```

- O interesse prático da aplicação da operação de Restrição sobre a Relação resultante de um Produto Cartesiano,
  - levou à definição da operação de Junção (*Join*)

Junção

- Qual o material fornecido por cada fornecedor ?
  - FORNECE $\bowtie_{\text{codMaterial = numeroMaterial}}$ MATERIAL

SELECT *

FROM FORNECE INNER JOIN MATERIAL

ON (codMaterial = numeroMaterial)

codFornecedor	codMaterial	numeroMaterial	nome
1	001	001	Parafuso
2	001	001	Parafuso
2	002	002	Roda Dentada
3	002	002	Roda Dentada

- A sintaxe geral de uma Junção é a seguinte:

… FROM <tabela1> [NATURAL] [<tipoJunção>] JOIN <tabela2>

[ON <condição>] ...

  - <tipoJunção> ::= INNER | <junçãoExterna> [OUTER]
  - <junçãoExterna> ::= LEFT | RIGHT | FULL

Inner join implicito -> SELECT * FROM FORNECE, MATERIAL **WHERE** codMaterial = numeroMaterial
Inner join explicito -> SELECT * FROM FORNECE **INNER JOIN** MATERIAL ON (codMaterial = numeroMaterial)

Ambiguidade na identificação dos Atributos

- Pode existir ambiguidade nas operações quando,
  - dois (ou mais) Atributos têm o mesmo nome em Relações diferentes
- A ambiguidade é eliminada com a qualificação do Atributo,
  - colocando o nome da Relação como prefixo do nome do Atributo
- Considere-se o seguinte Esquema Relacional
  - FORNECE(codFornecedor, codMaterial)
  - MATERIAL(codMaterial , nome)
- Qual o código e nome dos materiais fornecidos por cada fornecedor?
  - $\pi_{3, 4, 1}$ (FORNECE $\bowtie_{2 = 1}$ MATERIAL)

```
SELECT FORNECE.codMaterial, nome, codFornecedor
    FROM FORNECE, MATERIAL
    WHERE FORNECE.codMaterial = MATERIAL.codMaterial
```

Ambiguidade na identificação dos Atributos (cont.)

- Pode também existir ambiguidade nas operações quando,
  - a mesma Relação é referida mais do que uma vez numa operação
  - a Relação na cláusula FROM é o resultado de um SELECT
- A ambiguidade é eliminada atribuindo um pseudónimo à Relação
- Considere-se o seguinte Esquema Relacional
  - EMPREGADO(numBI, primNome, ultNome, numBIChefe)
- Para cada empregado, pretende-se saber o seu primeiro e último nome e também o primeiro e último nome do seu chefe.
  - $\pi_{2, 3, 6, 7}$ (EMPREGADO $\bowtie_{4 = 1}$ EMPREGADO)

```
SELECT EMP.primNome, EMP.ultNome, CHEFE.primNome, CHEFE.ultNome
    FROM EMPREGADO AS EMP, EMPREGADO AS CHEFE
    WHERE EMP.numBIChefe = CHEFE.numBI
```

Junção, Projecção e Restrição

FORNECE

codFornecedor	codMaterial
1	001
2	001
2	002
3	002

MATERIAL

numeroMaterial	nome
001	Parafuso
002	Roda Dentada

- Qual o nome dos materiais fornecidos pelo fornecedor de código 1 ?
  - $\pi_{\text{nome}} (\sigma_{\text{codFornecedor} = 1} (\text{FORNECE} \bowtie_{\text{codMaterial} = \text{numeroMaterial}} \text{MATERIAL}))$

```
SELECT M.nome AS nomeDoMaterial
    FROM FORNECE AS F INNER JOIN MATERIAL AS M
        ON ( F.codMaterial = M.numeroMaterial )
    WHERE F.codFornecedor = 1
```

nomeDoMaterial
Parafuso

- Notar que também às colunas se podem atribuir pseudónimos
- Por vezes os pseudónimos são usados para melhorar a legibilidade

Select geradores de tabelas temporárias

- Um select gera uma relação que pode ser utilizado como uma tabela normal

```
SELECT F.nomeForn
    FROM ( SELECT *
                FROM FORNECEDOR
                WHERE localidade = 'Porto' ) AS F
```

Fornecedor

nomeForn	telefone	localidade
Continentix	222111333	Lisboa
Jumbix	222222111	Lisboa
Feira Novix	222333444	Porto
Pingo Docix	222555111	Porto

F

nomeForn	telefone	localidade
Feira Novix	222333444	Porto
Pingo Docix	222555111	Porto

nomeForn
Feira Novix
Pingo Docix

Junção, Projecção e Restrição (cont.)

- Outra solução para a questão anterior agora poderia ser:
  - $\pi_{\text{nome}}((\sigma_{\text{codFornecedor} = 1}(\text{FORNECE})) \bowtie_{\text{codMaterial} = \text{numeroMaterial}} \text{MATERIAL})$

```
SELECT M.nome
   FROM ( SELECT *
          FROM FORNECE
          WHERE codFornecedor = 1 ) AS F
   INNER JOIN
          MATERIAL AS M
   ON ( F.codMaterial = M.numeroMaterial )
```

- Nesta solução a operação de Restrição é efectuada antes da Junção
  - o conjunto de tuplos a considerar na Junção é menor do que na solução anterior
  - contudo os SGBD já fazem estas optimizações de forma automática

Não Equi-Junção

- A condição de Junção pode não impor igualdade entre duas colunas

NOTA

numAluno	disciplina	nota
111	Economia	15
222	Inglês	12
111	Álgebra	17
333	Estatistica	13
222	Biologa	16

ESCALA

notaMax	notaMin	classif
9	0	Reprovado
13	10	Suficiente
15	14	Bom
17	16	Muito Bom
20	18	Excelente

- Qual a classificação (classif) do aluno 222 nas diversas disciplinas ?
  - $\pi_{1, 2, 6}$ ($\sigma_{1 = 222}$ (NOTA $\bowtie_{3 >= 2 \wedge 3 <= 1}$ ESCALA))

numAluno	disciplina	classif
222	Inglês	Suficiente
222	Biologa	Muito Bom

```
SELECT numAluno, disciplina, classif
    FROM NOTA AS N INNER JOIN ESCALA AS E
        ON ( N.nota BETWEEN E.notaMin AND E.notaMax )
    WHERE N.numAluno = 222
```

- O operador BETWEEN é apenas uma forma simplificada de escrever
  - (nota >= notaMin AND nota <= notaMax)

Junções Múltiplas

- Considere-se o seguinte Esquema Relacional
  - CATEGORIA(codCat, nomeCat, salarioBase)
  - DEPARTAMENTO(codDep, nomeDep, localizacao)
  - EMPREGADO(cod_Emp, nomeEmp, codCat, codDep, codEmpChefe)

- Para cada categoria, pretende-se saber qual o nome dos empregados, salário base e respectivo nome do departamento
  - $\pi_{2, 5, 3, 10}$ ((CATEGORIA $\bowtie_{1 = 3}$ EMPREGADO) $\bowtie_{7 = 1}$ DEPARTAMENTO)

```
SELECT C.nomeCat, E.nomeEmp, C.salarioBase, D.nomeDep
FROM
    (  CATEGORIA AS C INNER JOIN EMPREGADO AS E
       ON ( C.codCat = E.codCat )  )  INNER JOIN  DEPARTAMENTO AS D
          ON ( E.codDep = D.codDep )
```

Junções Múltiplas (cont.)

- Outro modo de escrever a mesma interrogação é

```
SELECT nomeCat, nomeEmp, C.salarioBase, nomeDep
    FROM CATEGORIA AS C , EMPREGADO AS E, DEPARTAMENTO AS D
    WHERE C.codCat = E.codCat AND E.codDep = D.codDep
```

- Neste caso a diferença será essencialmente sintáctica
  - um interpretador de SQL poderá construir o mesmo plano de execução para a interrogação escrita de um ou de outro modo

Natural join

- Uma Junção Natural (Natural Join) compara todas as colunas, que têm o mesmo nome, nas duas tabelas
- A tabela resultante apenas contém uma coluna por cada par de colunas comparadas

- Expressão relacional de uma junção natural:
  - CATEGORIA ⋈ EMPREGADO

- Comando SQL com um natural join:

```
SELECT *
  FROM EMPREGADO NATURAL JOIN DEPARTAMENTO
```

- Esquema de Relação resultante
  - ???(cod_Emp, nomeEmp, codCat, codDep, codEmpChefe, nomeDep, localizacao)

CATEGORIA(codCat, nomeCat, salarioBase)
DEPARTAMENTO(codDep, nomeDep, localizacao)
EMPREGADO(cod_Emp, nomeEmp, codCat, codDep, codEmpChefe)

Junção Externa (*Outer Join*)

CATEGORIA(codCat, nomeCat, salarioBase)
DEPARTAMENTO(codDep, nomeDep, localizacao)
EMPREGADO(cod_Emp, nomeEmp, codCat, codDep, codEmpChefe)

- Quais os empregados e categorias existentes e para cada empregado quais as categorias superiores à sua ?
  - EMPREGADO $\bowtie_{3 < 1}$ CATEGORIA

```
SELECT *
   FROM EMPREGADO EMP FULL OUTER JOIN CATEGORIA CAT
      ON ( EMP.codCat < CAT.codCat )
```

De notar a omissão de **AS**

Junção Externa (cont.)

- Se apenas estiverem disponíveis as operação de Junção Externa à Esquerda e à Direita, a interrogação anterior pode ser escrita,
  - (EMPREGADO ⟕ $_{3<1}$CATEGORIA) ∪ (EMPREGADO ⟖ $_{3<1}$CATEGORIA)

```
SELECT *
    FROM EMPREGADO EMP LEFT OUTER JOIN CATEGORIA CAT
        ON ( EMP.codCat < CAT.codCat )
UNION
SELECT *
    FROM EMPREGADO EMP RIGHT OUTER JOIN CATEGORIA CAT
        ON ( EMP.codCat < CAT.codCat )
```

Junção Externa (cont. 1)

CATEGORIA(codCat, nomeCat, salarioBase)
DEPARTAMENTO(codDep, nomeDep, localizacao)
EMPREGADO(cod_Emp, nomeEmp, codCat, codDep, codEmpChefe)

- Quais os nomes de todos departamentos existentes e para cada um qual o código e nome dos empregados que lá trabalham ? Notar que devem ser apresentados todos os departamentos, mesmo os que não têm empregados.
  - $\pi_{2, 4, 5}$ (DEPARTAMENTO $\overset{\bullet}{\bowtie}_{1 = 4}$ EMPREGADO))

```
SELECT DEP.nomeDep, codEmp, EMP.nomeEmp
    FROM DEPARTAMENTO DEP LEFT OUTER JOIN EMPREGADO EMP
        ON ( DEP.codDep = EMP.codDep )
```

Funções de agregação

Funções de Agregação

- Podem ser usadas na cláusula SELECT ou HAVING (que será posteriormente apresentada)
  - aplicam-se sobre um grupo de linhas

- MAX(<expressão>)
  - valor máximo da expressão, para as linhas seleccionadas. O resultado não inclui tuplos duplicados.

- MIN(<expressão>)
  - valor mínimo da expressão, para as linhas seleccionadas. O resultado não inclui tuplos duplicados.

- <expressão>
  - pode incluir operadores aritméticos: +, - ,*, / e os ()

Funções de Agregação (cont.)

- COUNT(*)
  - número total de linhas
- COUNT([DISTINCT] <coluna>)
  - número de linhas excluindo as que, para a coluna indicada, têm valor NULL. Caso se use DISTINCT não se consideram valores duplicados
- SUM([DISTINCT] <expressão>)
  - somatório dos valores [diferentes] da expressão
- AVG([DISTINCT] <expressão>)
  - média (SUM / COUNT) dos valores [diferentes] da expressão
- SUM e AVG apenas se aplicam a valores numéricos

Funções de Agregação (cont. 1)

CATEGORIA(codCat, nomeCat, salarioBase)
DEPARTAMENTO(codDep, nomeDep, localizacao)
EMPREGADO(cod_Emp, nomeEmp, codCat, codDep, codEmpChefe)

- Qual o nome do empregado que aparecerá em primeiro lugar numa ordenação lexicográfica crescente ?
  - $\mathcal{F}_{\text{MIN nome}}$ (EMPREGADO)

```
SELECT MIN( nomeEmp )
FROM EMPREGADO
```

```
SELECT AVG(salarioEfectivo*10)*10  FROM  EMPREGADO
```

Funções de Agregação (cont. 2)

- Qual o total de nomes diferentes dos empregados em 'Lisboa' ?
  - $\Im_{\text{COUNT 5}}$ (($\sigma_{\text{localizacao = 'Lisboa'}}$ (DEPARTAMENTO)) $\bowtie$ EMPREGADO)

```
SELECT COUNT( DISTINCT E.nomeEmp )
    FROM DEPARTAMENTO AS D INNER JOIN EMPREGADO AS E
            ON ( D.codDep = E.codDep )
    WHERE localizacao = 'Lisboa'
```

- Qual o valor total dos salários base a pagar aos empregados do departamento 1 e qual o valor de IRS (calculado a 22%)?
  - $\Im_{\text{SUM salarioBase, SUM (salarioBase*0.17)}}$ (
    ($\sigma_{\text{codDep = 1}}$ (EMPREGADO)) $\bowtie_{3=1}$ CATEGORIA)

```
SELECT SUM( C.salarioBase ), SUM( C.salarioBase * 0.22 ) AS IRS
    FROM EMPREGADO AS E INNER JOIN CATEGORIA AS C
            ON ( E.codCat = C.codCat )
     WHERE codDep = 1
```

CATEGORIA(codCat, nomeCat, salarioBase)
DEPARTAMENTO(codDep, nomeDep, localizacao)
EMPREGADO(cod_Emp, nomeEmp, codCat, codDep, codEmpChefe)

Sub-Interrogação (*subquery*)

CATEGORIA(codCat, **nome**, salarioBase)
DEPARTAMENTO(codDep, **nome**, localizacao)
EMPREGADO(cod_Emp, **nome**, codCat, codDep, codEmpChefe)

- Qual o nome dos empregados que trabalham no mesmo departamento que o(s) empregado(s) com nome 'Carlos Castro' ?

```
SELECT DISTINCT E1.nome
FROM   EMPREGADO AS E1 INNER JOIN EMPREGADO AS E2
           ON ( E1.codDep = E2.codDep )
WHERE E1.nome <> 'Carlos Castro' AND E2.nome = 'Carlos Castro'
```

- Outra solução consiste em considerar duas sub-interrogações:
  - Qual o código do departamento do(s) empregado(s) 'Carlos Castro' ?
  - Qual o nome dos empregados do departamento, com o código obtido na interrogação anterior, mas que não são 'Carlos Castro' ?

Sub-Interrogação (cont.)

- Qual o código do departamento do(s) empregado(s) 'Carlos Castro' ?

```
SELECT DISTINCT E2.codDep
    FROM EMPREGADO AS E2
    WHERE E2.nome = 'Carlos Castro‘
```

- Qual o nome dos empregados do(s) departamento(s) com os códigos obtidos na interrogação anterior, mas que não são ‘Carlos Castro’ ?
  - assumindo que existem cinco empregados com o nome ‘Carlos Castro’ e que que três deles estão no departamento 4, um está no 7 e outro no 11

```
SELECT DISTINCT E1.nome
    FROM EMPREGADO AS E1
    WHERE E1.nome <> 'Carlos Castro' AND
                E1.codDep IN ( 4, 7, 11 )
```

Sub-Interrogação (cont. 1)

- Resposta à questão inicial:
  - Qual o nome dos empregados que trabalham no mesmo departamento que o(s) empregado(s) com nome 'Carlos Castro' ?

- Obtida através da composição das duas interrogações numa única interrogação:

```
SELECT DISTINCT E1.nome
    FROM EMPREGADO AS E1
    WHERE E1.nome <> 'Carlos Castro' AND
                E1.codDep IN
                        ( SELECT DISTINCT E2.codDep
                            FROM EMPREGADO AS E2
                            WHERE E2.nome = 'Carlos Castro' )
```

- Sub-interrogação (interrogação interior): "(SELECT … E2.codDep ..."

Sub-Interrogação Não-Correlacionada

- Numa Sub-Interrogação Não-Correlacionada a interrogação interior não necessita dos valores da interrogação exterior

- Quais os códigos e nomes das categorias com menor salário base ?

```
SELECT C.codCat, C.nome
    FROM CATEGORIA AS C
    WHERE C.salarioBase =
            ( SELECT MIN( D.salarioBase )
                FROM CATEGORIA AS D )
```

- A interrogação interior (SELECT MIN(…) não depende da exterior
  - a interrogação interior é executada em primeiro lugar e apenas uma vez
  - a relação devolvida na interrogação interior permite resolver a exterior

Sub-Interrogação Correlacionada

- Numa Sub-Interrogação Correlacionada a interrogação interior necessita de valores da interrogação exterior

- Quais as categorias cujo salário base é inferior a metade do valor médio dos salários efectivos dos empregados dessas categorias ?

CATEGORIA(codCat, nome, salarioBase)
EMPREGADO(codEmp, nome, salarioEfectivo, codCat, codDep)

```
SELECT C.*
    FROM CATEGORIA AS C
    WHERE C.salarioBase <
            ( ( SELECT AVG( E.salarioEfectivo )
                FROM EMPREGADO AS E
                WHERE E.codCat = C.codCat )  /  2 )
```

- A interrogação interior (SELECT AVG(…) depende da exterior
  - a informação da interrogação exterior é passada à interior *
  - para cada linha da interrogação exterior é executada a interior

* Devido ao WHERE E.codCat = C.codCat, que compara com um valor da interrogação externa

Cláusula WHERE

- Foi anteriormente apresentada a sintaxe geral de uma questão SQL

  SELECT [DISTINCT] <colunas> | *

  FROM <tabelas>

  [WHERE <condição>]

- <condição>
  - consiste numa colecção de “Predicados”

- Cada “Predicado”,
  - pode envolver qualquer composição de sub-expressões do mesmo tipo,
  - combinada com os operadores lógicos AND, OR, NOT e parêntesis

- Cada “Predicado”,
  - quando avaliado produz um valor lógico verdadeiro ou falso, para o tuplo considerado

Predicados

- Os Predicados podem ser utilizados num contexto estático, sendo avaliados com base em valores constantes.
  - ... WHERE E1.codDep IN (4, 7, 11) ...

- Alguns Predicados também ser avaliados com base em valores dinâmicos, a retirar da base de dados
  - … WHERE D.codDep IN (SELECT E2.codDep FROM EMPREGADO AS E2)
  - neste caso é utilizada um “Sub-Interrogação” (Subquery)

- As “Sub-Interrogações” podem-se usar em Predicados:
  - Comparação e BETWEEN
  - IN
  - ALL, ANY
  - EXISTS

Predicados (cont.)

- LIKE: pesquisa cadeias de caracteres com determinado padrão
- comparação (=, <>, >, <, >=, <=) e BETWEEN
  - WHERE nomeEmp **=** 'Manuel Silva'
  - WHERE codEmp **BETWEEN** 1 AND 5
- IN
  - WHERE codCat **IN** (1, 2)
- ANY (em pelo menos um valor *), ALL (em todos os valores *)
  - WHERE salarioBase > **ANY** (SELECT …)
- EXISTS (se existe pelo menos uma linha *)
  - WHERE **EXISTS** (SELECT …)
- Teste de valor nulo
  - WHERE comissao **IS** NULL

* da subquery

O conceito de valor *NULL*

- O valor *NULL* permite lidar com situações onde, para determinado tuplo, o valor de um atributo seja indefinido ou desconhecido.
- Situações de utilização do valor *NULL*:
  - o atributo não é aplicável para determinado tuplo;
  - o atributo tem um valor desconhecido para determinado tuplo;
  - o atributo tem valor conhecido mas o valor está ausente nesse instante, ou seja, ainda não foi registado na Base de Dados.
- Características dos valores *NULL*:
  - Independente do domínio - inteiro, real, caracter, data, etc.
  - Não comparáveis entre si - uma expressão com um operador de comparação será avaliada como *FALSE*, se algum dos seus operandos tiver o valor *NULL*.
  - Existe função (ISNULL) para substituir o valor *NULL* por outro valor.
  - Existe directiva (IS NULL) para encontrar valores *NULL* numa interrogação (com colunas definidas como permitindo valores *NULL*).

```
SELECT ISNULL(codCat, 1)  FROM EMPREGADO WHERE codDep = 3
SELECT * FROM EMPREGADO WHERE salarioEfectivo IS NULL;
```

LIKE

- Pesquisa de cadeias de caracteres que tenham determinado padrão
- <expressão> [NOT] LIKE <padrão> [ESCAPE <caracterExcepção>]
  - expressão: deve ser do tipo cadeia de caracteres (string)
  - padrão: pode incluir meta-caracteres, que não sendo precedidos do caracter de excepção, têm o seguinte significado,
    - '%' qualquer sequência de zero ou mais caracteres
    - '_' um único qualquer caracter
    - [G-J] ou [GHIJ] qualquer caracter do intervalo (G a J neste exemplo)
    - [^G-J] ou [^GHIJ] qualquer caracter não pertencente ao intervalo
    - caracterExcepção ocorre em padrão anulando tratamento especial
- Quais as categorias cujo primeiro caracter do nome não pertence ao alfabeto (a A, … z Z) e que inclui um *underscore* (_) ?

  SELECT * FROM CATEGORIA

  WHERE nome LIKE '[^A-Za-z]%+_%' ESCAPE '+'

O carácter '_' é precedido do carácter de escape '+', pode-se definir um qualquer carácter de escape

IN

- Verifica se um valor está contido numa coluna de uma tabela, ou numa lista de valores
- <construtor de linha> [NOT] IN
  ({sub-interrogação | lista de expressões escalares})
- Quais as categorias dos empregados com maior salário efectivo ?

CATEGORIA(codCat, nome, salarioBase)
EMPREGADO(codEmp, nome, salarioEfectivo, codCat, codDep)

```
SELECT C.*
    FROM CATEGORIA AS C
    WHERE C.codCat IN
                ( SELECT E.codCat
                    FROM EMPREGADO AS E
                    WHERE E.salarioEfectivo =
                            ( SELECT MAX( F.salarioEfectivo )
                                FROM EMPREGADO AS F ) )
```

IN (cont.)

- Quais as categorias dos empregados com maior salário efectivo ?
  - sem utilizar o Predicado IN, temos outra solução,
  - CATEGORIA$\bowtie_{1=4}$ (EMPREGADO$\bowtie_{3=1}$($\Im$ $_{\text{MAX salarioEfectivo}}$ (EMPREGADO)))

```
SELECT C.*
    FROM CATEGORIA AS C INNER JOIN
                (
                EMPREGADO AS E INNER JOIN
                        ( SELECT MAX( salarioEfectivo )
                            FROM EMPREGADO ) AS E1( salarioEfectivoMax )
                ON ( E.salarioEfectivo = E1.salarioEfectivoMax )
                )
            ON ( C.codCat = E.codCat )
```

E1(salarioEfectivoMax) → declaração do nome da tabela e da coluna

ANY, ALL

- Verifica se todas (ALL) ou algumas (ANY) linhas, da subquery, têm um atributo que obedece a uma expressão envolvendo operadores relacionais
- <construtor de linha> <comparação> {ALL | ANY} (sub-interrogação)
- Qual o nome dos empregados cujo salário efectivo é superior ao de alguns empregados da mesma categoria ?

```
SELECT E.nome
    FROM EMPREGADO AS E
    WHERE E.salarioEfectivo > ANY
                ( SELECT E1.salarioEfectivo
                    FROM EMPREGADO E1
                    WHERE E.codCat = E1.codCat )
```

  - substituindo ANY por ALL teríamos como resultado uma Relação vazia

ALL

- Qual o nome dos empregados cujo salário efectivo é superior ao de todos os empregados de departamentos localizados em 'Lisboa' ?

```
SELECT E.nome
    FROM EMPREGADO AS E
    WHERE E.salarioEfectivo > ALL
        ( SELECT E1.salarioEfectivo
            FROM EMPREGADO AS E1 INNER JOIN DEPARTAMENTO AS D
                ON ( E1.codDep = D.codDep )
            WHERE D.localizacao = 'Lisboa' )
```

- Sem utilizar o Predicado ALL, temos outra solução,
  - "… E.salarioEfectivo > (SELECT MAX(E1.salarioEfectivo) …"

CATEGORIA(codCat, nome, salarioBase, localização)
EMPREGADO(codEmp, nome, salarioEfectivo, codCat, codDep)

EXISTS, NOT EXISTS

- Verifica a existência de, pelo menos, uma linha numa tabela
- [NOT] EXISTS (sub-interrogação)
- Quais os departamentos que têm pelo menos um empregado ?

```
SELECT D.*
    FROM DEPARTAMENTO AS D
    WHERE EXISTS
            ( SELECT *
                FROM EMPREGADO AS E
               WHERE E.codDep = D.codDep )
```

- A condição do WHERE é verdadeira se,
  - o resultado da sub-interrogação for não vazio.

EXISTS (cont.)

- A questão anterior pode-se responder com operador de intersecção
- Quais os departamentos que têm pelo menos um empregado ?
  - DEPARTAMENTO $\cap$ ($\pi_{1,2,3}$ (DEPARTAMENTO $\bowtie_{1=5}$ EMPREGADO))
- EXISTS pode ser usado para escrever o operador INTERSECT

```
SELECT D.*
    FROM DEPARTAMENTO AS D
    WHERE EXISTS
        ( SELECT *
            FROM DEPARTAMENTO AS D1 INNER JOIN EMPREGADO AS E
                ON ( D1.codDep = E.codDep )
            WHERE D.codDep = D1.codDep )
```

DEPARTAMENTO(codDep, nome, localizacao)
EMPREGADO(codEmp, nome, salarioEfectivo, codCat, codDep)

EXISTS (cont. 1)

- Quais os departamentos que têm pelo menos um empregado ?
  - sem utilizar o Predicado EXISTS, temos as seguintes soluções:

```
SELECT D.*
    FROM DEPARTAMENTO AS D
    WHERE ( SELECT COUNT( codEmp )
                FROM EMPREGADO AS E
                WHERE E.codDep = D.codDep ) > 0

SELECT DISTINCT D.*
    FROM DEPARTAMENTO AS D INNER JOIN EMPREGADO AS E
        ON ( D.codDep = E.codDep )
```

NOT EXISTS

- Qual o código e nome das categorias que não têm empregados ?

```
SELECT C.codCat, C. nome
    FROM CATEGORIA AS C
    WHERE NOT EXISTS
        ( SELECT *
            FROM EMPREGADO AS E
            WHERE C.codCat = E.codCat )
```

- A condição do WHERE é verdadeira se,
  - o resultado da sub-interrogação for vazio.

- Sem utilizar o Predicado NOT EXISTS, temos outra solução,
  - "… WHERE 0 = (SELECT COUNT(E.codCat) …"

Comparação (=)

- Quais as categorias com maior salário base ?

```
SELECT C.codCat
    FROM CATEGORIA AS C
    WHERE C.salarioBase =
        ( SELECT MAX( C1.salarioBase )
            FROM CATEGORIA AS C1 )
```

- Qual o código e nome dos empregados com maior salário base ?

```
SELECT E.codEmp, E.nome
    FROM EMPREGADO AS E INNER JOIN CATEGORIA AS C
      ON ( E.codCat = C.codcat)
    WHERE C.salarioBase =
        ( SELECT MAX( C1.salarioBase )
            FROM CATEGORIA AS C1 )
```

DEPARTAMENTO(codDep, nome, localizacao)
CATEGORIA(codCat, nome, salarioBase)
EMPREGADO(codEmp, nome, salarioEfectivo, codCat, codDep)

Comparação (>)

- Considere-se o seguinte Esquema Relacional
  - ALUNO(numAluno, nome, dataNascimento, localNascimento)
- Pretende-se uma lista onde, para cada local indique todos os alunos mais novos do que o mais velho que lá nasceu ?

```
SELECT A1.localNascimento, A1.numAluno, A1.nome
    FROM ALUNO AS A1
    WHERE A1.dataNascimento >
                ( SELECT MIN( A2.dataNascimento )
                        FROM ALUNO AS A2
                        WHERE A1.localNascimento = A2.localNascimento )
```

Operação de Divisão

- Quais os códigos dos departamentos que têm empregados de todas as categorias ?

EMPREGADO

codEmp	nomeEmp	codCat	codDep
1	António Abreu	1	1
2	Bernardo Bento	1	**2**
3	Carlos Castro	3	3
4	Manuel Matos	2	**2**
5	Manuel Matos	3	**2**

CATEGORIA

codCat	designacao	salarioBase
1	CategoriaA	300
2	CategoriaB	250
3	CategoriaC	160

- O resultado obtido pode-se obter efectuando uma operação de Divisão entre EMPREGADO e CATEGORIA
  - $\pi_{3,4}$ (EMPREGADO) ÷ π_{1} (CATEGORIA)

codDep
2

Operação de Divisão (cont.)

- Quais os códigos dos departamentos que têm empregados de todas as categorias ?
- Ou, colocando a questão de outro modo:
  - Quais os códigos dos departamentos para os quais,
  - qualquer que seja a categoria,
  - existe algum empregado desse departamento e dessa categoria
- Ou, utilizando uma notação simbólica:

  codigoDepartamento : $\forall$ categoria $\in$ CATEGORIA,

  ($\exists$ empregado :

  EMPREGADO.codDep = codigoDepartamento

  AND EMPREGADO.codCat = CATEGORIA.codCat **)**
- Vamos designar a expressão que, está escrita entre parêntesis, por:
  - p(empregado)

Operação de Divisão (cont. 1)

- Sabendo que,
  - $\forall$ x : p(x) $\Leftrightarrow$ $\neg \exists$ x : $\neg$ p(x)

- Tem-se,

  codigoDepartamento : $\neg \exists$ categoria $\in$ CATEGORIA,

  ($\neg$ p(codigoDepartamento) **)**

- Ou seja,

  codigoDepartamento : $\neg \exists$ categoria $\in$ CATEGORIA,

  ($\neg \exists$ empregado :

  EMPREGADO.codDep = codigoDepartamento

  AND EMPREGADO.codCat = CATEGORIA.codCat **)**

  Os departamentos (codDep) onde não exista uma categoria que não tenha um empregado.

Operação de Divisão (cont. 2)

- Pretende-se a informação completa sobre os departamentos que têm empregados de todas as categorias ?
  - $DEPARTAMENTO \bowtie_{1=1} (\pi_{3,4} (EMPREGADO) \div \pi_{1} (CATEGORIA))$

```
SELECT D.*
    FROM DEPARTAMENTO AS D
    WHERE NOT EXISTS
                ( SELECT *
                    FROM CATEGORIA AS C
                    WHERE NOT EXISTS
                        ( SELECT *
                            FROM EMPREGADO AS E
                            WHERE  E.codDep = D.codDep
                                AND
                                E.codCat = C.codCat ) )
```

Comando Select – order by, group by e having

Select …
from …
where …
group by…
having …
order by …

Comando SELECT

- A sintaxe de uma interrogação SQL inclui as seguintes cláusulas:

```
SELECT [DISTINCT] <colunas> | *
   FROM <tabela>, ...
   [WHERE <condição>]
   [GROUP BY <expressão de agrupamento>]
   [HAVING <condição>]
   [ORDER BY <expressão de ordenação> [ASC | DESC], … ]
```

ORDER BY

- Ordena a relação resultante, por ordem crescente ou decrescente de uma ou mais expressões de ordenação
  - é sempre a última cláusula a ser especificada

```
SELECT [DISTINCT] <colunas> | *
    FROM <tabela>, ...
   [WHERE <condição>]
   [ORDER BY <expressão de ordenação> [ASC | DESC], … ]
```

- ASC: ordenação ascendente (*ascending*), valor por omissão
- DESC: ordenação descendente (*descending*)

ORDER BY (cont.)

- Pretende-se toda a informação da tabela de empregado, ordenada lexicograficamente pelo nome do empregado, de modo crescente

```
SELECT *
   FROM EMPREGADO
   ORDER BY nome ASC
```

- Pretende-se a mesma informação que a apresentada no exemplo anterior, mas neste caso, os nomes repetidos devem aparecer ordenados de forma decrescente pelo código do empregado

```
SELECT *
   FROM EMPREGADO
   ORDER BY nome ASC, codEmp DESC
```

GROUP BY

- Produz uma linha de resultados por cada conjunto de linhas com o mesmo valor da expressão de agrupamento
  - Permite, por exemplo, aplicar funções de agregação a grupos de linhas de uma tabela.

SELECT [DISTINCT] <colunas> | *
FROM <tabela>, ...
[WHERE <condição>]
[GROUP BY <expressão de agrupamento>]

- A cláusula SELECT apenas pode conter:
  - colunas que apareçam na cláusula GROUP BY e,
  - funções de agregação aplicadas a quaisquer outras colunas

GROUP BY (cont.)

- Para cada código de departamento qual o seu salário base mínimo ?

```
SELECT E.codDep, MIN ( C.salarioBase ) AS salarioMinimo
    FROM EMPREGADO AS E INNER JOIN CATEGORIA AS C
        ON( E.codCat = C.codCat )
    GROUP BY E.codDep
```

- Agrupamento Múltiplo
  - Para cada código de departamento, qual o salário efectivo mínimo praticado para cada código de categoria ?

```
SELECT E.codDep, E.codCat, MIN( E.salarioEfectivo ) AS salarioMinimo
    FROM EMPREGADO AS E INNER JOIN CATEGORIA AS C
        ON( E.codCat = C.codCat )
    GROUP BY E.codDep, E.codCat
```

DEPARTAMENTO(codDep, nome, localizacao)
CATEGORIA(codCat, nome, salarioBase)
EMPREGADO(codEmp, nome, salarioEfectivo, codCat, codDep)

HAVING

- Aplica uma ou mais condições a cada grupo de linhas especificado pela cláusula GROUP BY
  - Se a cláusula GROUP BY não for utilizada, o grupo considerado é o de todas as linhas resultantes do SELECT
- A cláusula HAVING é usada para definir restrições a grupos, assim como a cláusula WHERE é usada para definir restrições a linhas.

  SELECT [DISTINCT] <colunas> | *

  FROM <tabela>, ...

  [WHERE <condição>]

  [GROUP BY <expressão de agrupamento>]

  [HAVING <condição>]
- Utilização das cláusulas HAVING e WHERE
  - a cláusula HAVING deve sempre conter funções de agregação
  - a cláusula WHERE nunca contém funções de agregação

HAVING (cont.)

- Pretende-se uma lista onde conste, para cada código de departamento o salário base mínimo nele praticado. Apenas se pretendem os departamentos com salário base médio superior a 50

```
SELECT E.codDep, MIN( C.salarioBase )
    FROM CATEGORIA AS C INNER JOIN EMPREGADO AS E
        ON( C.codCat = E.codCat )
    GROUP BY E.codDep
    HAVING AVG( C.salarioBase ) > 50
```

WHERE, GROUP BY, HAVING e ORDER BY

- Pretende-se uma lista onde conste, para cada código de departamento o valor do menor salário efectivo pago a empregados da categoria com código ‘C1’. Só se pretendem os departamentos para os quais esse menor salário efectivo seja superior ao menor salário base de todas as categorias. Os valores mais altos devem ser apresentados primeiro.

```
SELECT E.codDep, MIN( E.salarioEfectivo )
    FROM EMPREGADO AS E
    WHERE E.codCat = 'C1'
    GROUP BY E.codDep
    HAVING MIN( E.salarioEfectivo ) >
                    ( SELECT MIN( salarioBase ) FROM CATEGORIA )
    ORDER BY MIN( E.salarioEfectivo ) DESC
```