Bancos de Dados

8. Structured Query Language (SQL)

Introdução

- A padronização ao redor de SQL é uma das principais razões para o sucesso dos bancos de dados relacionais
  - A migração entre SGBDs é facilitada, embora existam muitas diferenças entre eles
  - O uso estrito do padrão preserva adequadamente essa possibilidade
  - Um SBD pode inclusive usar mais de um SGBD *diferente*, acessando-os apenas usando SQL padrão
- SQL foi padronizada em 1986 (SQL1) pelo ANSI e pela ISO
- Hoje está em vigor um padrão atualizado (SQL2), aprovado em 1992
- Está em estudos a SQL3, que trará extensões para BD orientados a objetos e outros avanços recentes

Introdução

- SQL é baseada na álgebra relacional e no cálculo relacional de tuplas
  - Álgebra relacional especifica uma *seqüência* de operações
  - Por outro lado, a SQL é uma linguagem *declarativa*, em que o usuário especifica apenas que resultado deseja, não como obtê-lo
  - Isso permite que o SQL tenha seu desempenho melhorado por estratégias de *otimização de consultas*, transparentes para o usuário, a cargo do SGBD
- SQL é ao mesmo tempo DDL e DML
  - Além disso, inclui recursos para especificar visões, segurança, autorizações, restrições de integridade e controle de transações

Exemplo do Cálculo Relacional de Tuplas

- No CRT, uma variável de tupla cobre toda uma relação, ou seja, a variável pode assumir, como valor, qualquer tupla da relação
- As expressões do CR são definidas como conjuntos
  - `{t | COND(t)}` → conjunto das tuplas t que atendem à condição COND
  - `{t | EMPREGADO(t) AND t.salario > 1000}` → conjunto dos dados (tupla completa) de empregado cujo salário seja maior que 1000.
    - `EMPREGADO(t)` define que a *relação de extensão* da tupla é a relação empregado
  - `{t.nome, t.dn | EMPREGADO(t) AND t.salario > 1000}` → retorna apenas o nome e a data de nascimento, dentre os atributos de empregado

Mais exemplos

- Recuperar os nomes e endereços de todos os empregados que trabalham para o departamento ‘Pesquisa’

$$\{t.NOME, t.ENDERECO \mid EMPREGADO(t) \_\_ AND \_\_ (\exists d)(DEPTO(d) \_\_ AND \_\_ d.NOMEDEP = 'Pesquisa' \_\_ AND \_\_ d.NUMERODEP = t.NUMDEP)\}$$

- Mais exemplos: seções 6.6.4, 6.6.5, 6.6.6

Comparação de Conceitos

(*genérica e parcial*)

ER	Conceitual	Entidade	Instância	Atributo
Relacional	Lógico	Relação	Tupla	Atributo
SQL	Físico	Tabela	Linha	Coluna

Catálogos e Esquemas

- A SQL2 tem o conceito de *esquema*, ausente da SQL1, que permite separar conjuntos de tabelas de acordo com a aplicação
- O esquema é identificado por um *nome*, tem um *identificador de autorização* (usuário *dono* do esquema), e também *descritores* para todos os *elementos* nele contidos (tabelas, restrições, visões, domínios, etc.)
- Criação:
  - CREATE SCHEMA <nome> AUTHORIZATION <usuário>
- Tabelas são referenciadas como <esquema.tabela>

Catálogos e Esquemas

- No SQL2, um catálogo é uma *coleção de esquemas*
- Além dos esquemas criados pelo usuário, o catálogo inclui um esquema especial (INFORMATION_SCHEMA) que fornece informação sobre todos os descritores de todos os esquemas contidos no catálogo para usuários autorizados
- Esquemas contidos no mesmo catálogo podem compartilhar alguns elementos, tais como descritores de domínio

Comandos DDL

Comando básico: CREATE TABLE

CREATE TABLE <nome>
(<nomeColuna> <tipoColuna> [<restriçãoAtributo>]
[, <nomeColuna> <tipoColuna> [<restriçãoAtributo>]]
[, restriçãoTabela [, restriçãoTabela]])

CREATE TABLE

- Tipos de dados e domínios
  - Numéricos: INTEGER, INT, SMALLINT, FLOAT, REAL, DOUBLE PRECISION, DECIMAL(i,j), DEC(i,j), NUMERIC(i,j), NUMBER(i,j)
  - Strings: CHAR(n), CHARACTER(n), VARCHAR(n), CHAR VARYING(n)
  - Strings de bits: BIT(n), BIT VARYING(n)
  - Data e hora: DATE, TIME, TIME(i), TIME WITH TIME ZONE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, INTERVAL
  - Domínio definido pelo usuário
    - CREATE DOMAIN <nome> AS <tipo>
    - CREATE DOMAIN <nome> AS <tipo>
      CHECK <condição> → vide exemplo pg. 234

CREATE TABLE

- Restrições de atributo
  - DEFAULT <valor>
    - Pode ser incluído na definição de um domínio
  - NOT NULL
  - CHECK (<expressão>)
- Restrições de chave e unicidade
  - PRIMARY KEY
  - UNIQUE
    - Chave alternativa (chave candidata no relacional)
- Restrições de integridade referenciais
  - FOREIGN KEY REFERENCES <tabela>(<atributo>)
    ON DELETE [SET NULL | DEFAULT | CASCADE]
    ON UPDATE [SET NULL | DEFAULT | CASCADE]

CREATE TABLE

- Pode-se dar nome a uma restrição
  - CONSTRAINT <nome> PRIMARY KEY (<atributo>)
  - CONSTRAINT <nome> FOREIGN KEY (<atributo>)
    REFERENCES <tabela>(<atributo>)
    ON DELETE [SET NULL | DEFAULT | CASCADE]
    ON UPDATE [SET NULL | DEFAULT | CASCADE]
  - CONSTRAINT <nome> UNIQUE (<atributo>)
  - CONSTRAINT <nome> CHECK (<expressão>)
    - Neste caso, CHECK pode estabelecer restrições envolvendo os valores de atributos da mesma tupla

Exemplo

```
CREATE TABLE empregado
(NOME VARCHAR(15) NOT NULL,
 CPF NUMBER(11) NOT NULL,
 DATANASC DATE,
 ENDERECO VARCHAR(50),
 SEXO CHAR(1) CHECK(SEXO = ‘M’ OR SEXO = ‘F’),
 SALARIO NUMBER(10,2),
 CPFSUPER NUMBER(11),
 NUMDEPTO NUMBER(5) NOT NULL,
 PRIMARY KEY(CPF),
 CONSTRAINT FK_EMP_EMP FOREIGN KEY(CPFSUPER) REFERENCES
   EMPREGADO(CPF),
 CONSTRAINT FK_EMP_DEP FOREIGN KEY(NUMDEPTO) REFERENCES
   DEPARTAMENTO(NUMDEP)
);
```

Exemplo

- Observar no exemplo:
  - A restrição PRIMARY KEY poderia ter sido colocada na linha do atributo CPF, já que a chave primária é simples
  - A restrição PRIMARY KEY poderia ter nome, se a linha correspondente fosse precedida de CONSTRAINT <nome>
  - Apenas o NUMDEPTO é NOT NULL, o que indica que o relacionamento entre empregado e departamento é total (todo empregado deve pertencer a um departamento)
  - CPFSUPER pode ser nulo, indicando que o empregado pode não ter supervisor
  - Foi colocada restrição sobre o atributo SEXO, referente a valores permitidos; poderia ser nomeada no final:
    - CONSTRAINT CK_SEXO CHECK (SEXO = ‘M’ OR SEXO = ‘F’)

Alteração de Esquema

- DROP SCHEMA <nome> [CASCADE | RESTRICT]
  - RESTRICT: só é excluído o esquema se ele não contiver nenhum elemento (default)
- DROP TABLE <nome> [CASCADE | RESTRICT]
  - RESTRICT: só será excluída a tabela se ela não for referenciada em nenhuma restrição nem em visões (default)

Alteração de Esquema

- ALTER TABLE
  - ALTER TABLE <nome> ADD <atributo> <tipo>
    - Não é permitido configurar NOT NULL neste caso
  - ALTER TABLE <nome> DROP <atributo> [CASCADE | RESTRICT]
  - ALTER TABLE <nome> ALTER <atributo> [SET DEFAULT <valor> | DROP DEFAULT]
  - ALTER TABLE <nome> ADD CONSTRAINT
  - ALTER TABLE <nome> DROP CONSTRAINT <nome> [CASCADE | RESTRICT]
    - Só se a restrição tiver recebido nome no CREATE TABLE

Consultas SQL

Comando básico: SELECT

SELECT <lista de atributos>
FROM <lista de tabelas>
WHERE <condição>

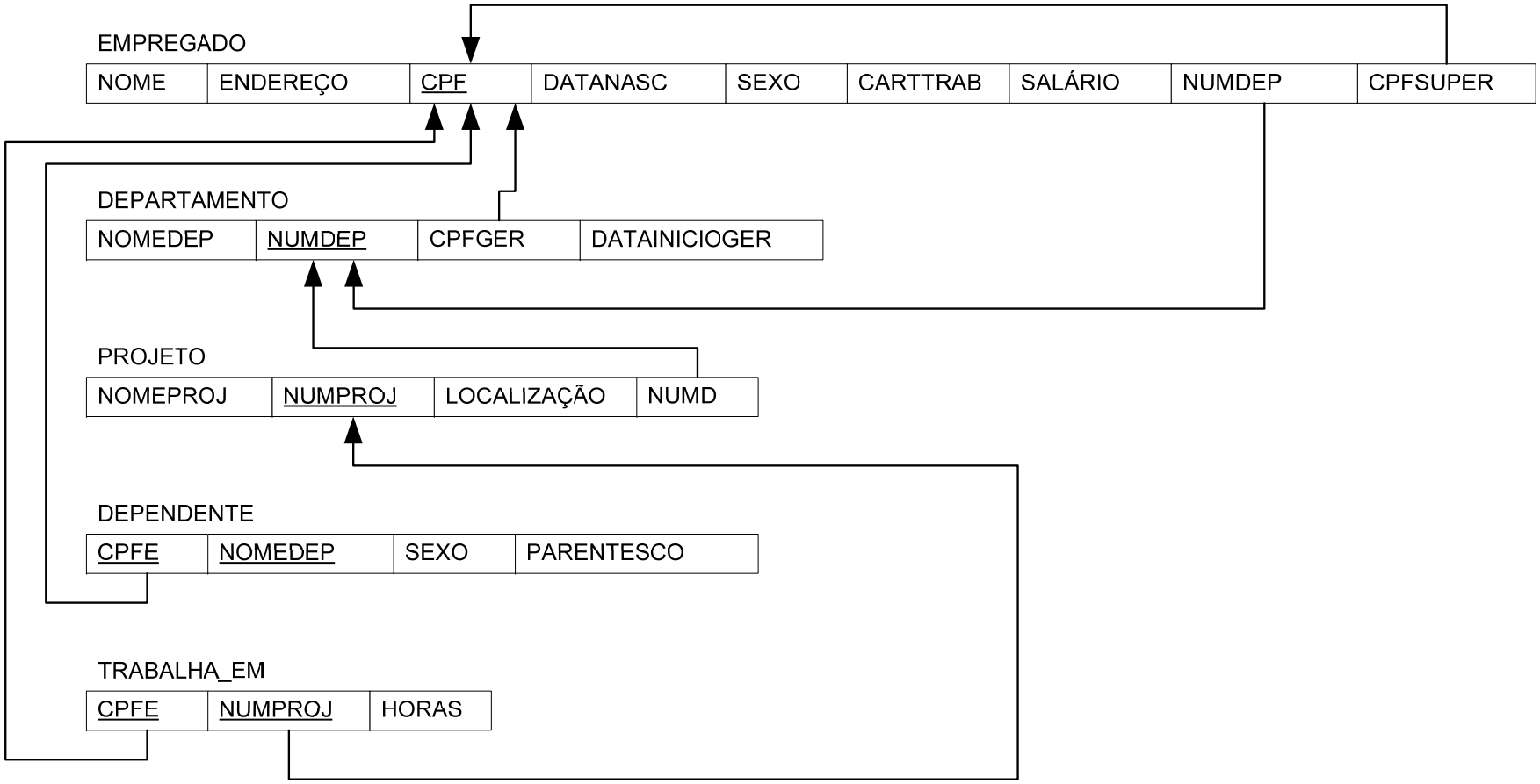
<condição> é um predicado booleano

SELECT

- SELECT com apenas uma tabela na lista FROM:
  - Par select-project da álgebra relacional
- SELECT com mais de uma tabela na lista FROM:
  - Consulta “select-project-join”
  - A condição WHERE deve incluir pelo menos uma *condição de junção*, que estabelece o relacionamento entre duas das tabelas listadas em FROM

Exemplo

- Os exemplos de SQL a seguir são baseados no projeto relacional a seguir, que é praticamente o mesmo que resultou do mapeamento apresentado em aulas anteriores (vide Mapeamento ER-Relacional)
- São adaptações dos exemplos apresentados em Elmasri&Navathe, cap. 8


EMPREGADO
NOME
ENDEREÇO
CPF
DATANASC
SEXO
CARTTRAB
SALÁRIO
NUMDEP
CPFSUPER
DEPARTAMENTO
NOMEDEP
NUMDEP
CPFGER
DATAINICIOGER
PROJETO
NOMEPROJ
NUMPROJ
LOCALIZAÇÃO
NUMD
DEPENDENTE
CPFE
NOMEDEP
SEXO
PARENTESCO
TRABALHA_EM
CPFE
NUMPROJ
HORAS

Q0 (select-project)

Recuperar a data de aniversário e o endereço do empregado cujo nome é ‘João da Silva’.

```
SELECT   datanasc, endereco
FROM     empregado
WHERE    nome = ‘João da Silva’
```

Q1 (select-project-join)

Recuperar o nome e o endereço de todos os empregados que trabalham no departamento cujo nome é ‘Pesquisa’.

```
SELECT   nome, endereco
FROM     empregado, departamento
WHERE    empregado.numdep =
         departamento.numdep
AND      nomedep = ‘Pesquisa’
```

SELECT

- Nomes de atributos ambíguos (*numdep* em Q1)
  - O mesmo nome pode ser usado para dois atributos que estejam em tabelas diferentes
  - Se ambos forem usados em uma consulta, deve-se prefixá-los com o nome da tabela: TABELA.ATRIBUTO
  - Pode ocorrer ambiguidade quando é necessário fazer referência à mesma tabela duas vezes
    - SELECT E.NOME, S.NOME
      FROM EMPREGADO AS E, EMPREGADO AS S
      WHERE E.CPFSUPER=S.CPF;
  - A cláusula AS pode ser usada para renomear todos os atributos de uma só vez:
    - EMPREGADO AS E(N, I, S, NSS, DN, END, SEX, SAL, NSSS, NUD)
    - AS é opcional ou mesmo não suportado em alguns SGBD (ex. Oracle)

Q2 (select-project-join)

Para cada projeto localizado em “Betim”, listar o nome do projeto, o número do departamento responsável, o nome do gerente do departamento, seu endereço e data de nascimento.

```
SELECT      numproj, numdep, nome, endereco, datanasc
FROM        projeto p, empregado e, departamento d
WHERE       numd = numdepto
AND         cpfger = cpf
AND         localizacao = ‘Betim’
```

Q8 (nomes ambíguos)

Para cada empregado, recuperar o nome e o nome do supervisor responsável por ele.

```
SELECT   e.nome, s.nome
FROM     empregado e, empregado s
WHERE    e.cpf = s.cpfger
```

Q9 (WHERE ausente)

Listar os CPFs e nomes de todos os empregados.

```
SELECT   cpf, nome
FROM     empregado
```

Default: WHERE true → todas as tuplas retornam

Q10 (WHERE ausente)

Para cada empregado, recuperar o nome e o nome do supervisor responsável por ele.

```
SELECT   nome, nomedep
FROM     empregado, departamento
```

Default: WHERE true → Não há condição de junção, portanto ocorre apenas o produto cartesiano.

Q1C (uso do asterisco)

Recuperar todos os dados de empregados do departamento 5

```
SELECT   *
FROM     empregado
WHERE    numdepto = 5
```

* Indica todos os atributos

Q1D (uso do asterisco)

Recuperar todos os dados de empregados do departamento ‘Pesquisa’

```
SELECT   *
FROM     empregado, departamento
WHERE    empregado.numdep =
         departamento.numdep
AND      nomedepto = ‘Pesquisa’
```

* Neste caso trará todos os atributos de empregado e de departamento, como resultado da junção

SELECT

- WHERE pode não ser especificada (default: TRUE)
  - `SELECT NOME FROM EMPREGADO`
  - Se existir mais de uma tabela na lista FROM, o resultado é o produto cartesiano das duas tabelas
    - `SELECT NOME, NOMEDEP FROM EMPREGADO, DEPENDENTE`
    - Equivale a produto cartesiano seguido de PROJECT na álgebra relacional
- A lista de atributos pode não ser especificada: usar *
  - `SELECT * FROM DEPARTAMENTO`

Q11 (ALL/DISTINCT)

Listar o salário de todos os empregados.

```
SELECT    ALL salario
FROM      empregado
```

Listar todos os diferentes valores de salários praticados na empresa.

```
SELECT    DISTINCT salario
FROM      empregado
```

Default: ALL (não precisa ser especificado)

SELECT

- A SQL não trata tabelas como conjuntos, mas sim como multiconjuntos
  - Tuplas duplicadas podem aparecer mais de uma vez numa tabela (desde que não exista chave primária) ou no resultado de uma consulta
- A eliminação de tuplas duplicadas não é realizada automaticamente porque:
  - Custa caro computacionalmente, pois requer ordenação
  - Pode ser do interesse do usuário ver as duplicatas
  - Pode interferir no resultado de operações de agregação
- SELECT DISTINCT / SELECT [ALL]
  - `SELECT DISTINCT SALARIO FROM EMPREGADO`

Q4 (operações de conjuntos)

Listar os números de projetos nos quais esteja envolvido o empregado ‘João da Silva’ como empregado ou como gerente responsável pelo departamento que controla o projeto.

```
(SELECT    DISTINCT numproj
FROM       projeto p, departamento d, empregado e
WHERE      p.numd = d.numdep
AND        d.cpfger = e.cpf
AND        e.nome = ‘João da Silva’)
UNION
(SELECT    DISTINCT numproj
FROM       projeto p, trabalha_em t, empregado e
WHERE      p.numproj = t.numproj
AND        t.cpfe = e.cpf
AND        e.nome = ‘João da Silva’)
```

Q4 (operações de conjuntos)

- Observação:
  - UNION, INTERSECT e EXCEPT retornam apenas tuplas distintas
  - Para evitar a eliminação de tuplas duplicadas, usar UNION ALL, INTERSECT ALL e EXCEPT ALL

SELECT

- A SQL incorpora diretamente algumas das operações de conjuntos da álgebra relacional
  - UNION (SQL1), INTERSECTION (SQL2), EXCEPT (SQL2)
- O resultado são *conjuntos de tuplas*, portanto tuplas duplicadas serão eliminadas
  - Obs: cuidado com o desempenho
  - UNION ALL, INTERSECT ALL e EXCEPT ALL anulam este efeito
- As operações são aplicáveis apenas sobre tabelas *compatíveis para união*, dentro do conceito da AR
  - As duas tabelas devem conter os mesmos atributos, na mesma ordem

Q12/Q13/Q14 (substrings, operadores aritméticos, LIKE)

```
SELECT      nome
FROM        empregado
WHERE       endereco LIKE ‘%Belo Horizonte%’

SELECT      nomeproj
FROM        projeto
WHERE       nomeproj LIKE ‘P_ _ _ _ _’

SELECT      nome, 1.1*salario AS novo_sal //obs. Rename de atributo
FROM        empregado e, departamento d
WHERE       e.numdep = d.numdep
AND         nomedep = ‘Pesquisa’

SELECT      *
FROM        empregado
WHERE       salario BETWEEN 1000 AND 3000
```

SELECT

- Operador LIKE: comparação parcial
  - % substitui qualquer número de caracteres e _ (underscore) substitui um caractere
  - Para usar % ou _ no padrão, digitar \% ou _
    - `SELECT NOME FROM EMPREGADO WHERE NOME LIKE '%Silva%';`
- Operações aritméticas
  - +, -, *, /
  - +, - servem para datas, horas e etc. também
- Strings
  - Concatenação: ||
- Comparação
  - >, <, >=, <=, =, <>
  - BETWEEN
    - ```
      SELECT NOME, SALARIO*1.1 FROM EMPREGADO
      WHERE SALARIO BETWEEN 1000 AND 2000
      ```
      ```

Q15 (ordenação)

Recuperar a lista de empregados e de departamentos e projetos em que trabalham, ordenando por nome de departamento, do empregado e do projeto

```
SELECT   nomedep, nome, nomeproj
FROM     departamento d, empregado e,
         projeto p, trabalha_em t
WHERE    d.numdep = e.numdep
AND      t.cpfe = e.cpf
AND      t.numproj = p.numproj
ORDER BY nomedep, nome, nomeproj
```

SELECT

- Ordenação
  - Cláusula ORDER BY
  - ORDER BY <atrib> [DESC | ASC] [, <atrib> [DESC | ASC]]
  - ASC é o default
  - ```
    SELECT *
    FROM EMPREGADO
    WHERE NUMDEP = 5
    ORDER BY SALARIO DESC;
    ```
- Observar que o retorno ordenado de uma consulta não depende da indexação da tabela
- A criação de índices é uma decisão de implementação, e inclusive não é parte (formal) da SQL
    ```

Consultas mais complexas

- NULL pode indicar
  - Valor desconhecido
  - Valor indisponível ou omitido
  - Valor não aplicável à tupla
- Comparações envolvendo NULL são diferenciadas:
  - Operadores especiais para comparar com a constante NULL
  - lógica de três valores

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Q18 (valor nulo)

Recuperar os nomes de todos os empregados que não têm supervisor.

```
SELECT      nome
FROM        empregado
WHERE       cpfsuper IS NULL
```

Observação: na junção, apenas combinações de tuplas em que a cláusula de junção for TRUE serão formadas; portanto, se a cláusula de junção for avaliada como FALSE ou UNKNOWN, a tupla não será selecionada. Exceção: OUTER JOIN

```
SELECT      e.nome, s.nome
FROM        empregado e, empregado s
WHERE       e.cpfsuper = s.cpf
// empregados que não têm supervisor não serão
   incluídos no resultado
```

SELECT

- Subqueries:
  - algumas operações exigem que dados sejam recuperados do banco e considerados na consulta
  - Subconsultas (subqueries) são blocos SELECT..FROM..WHERE inseridos na cláusula WHERE da consulta *externa*
  - Usar o operador de comparação IN

    SELECT <atribs>
    FROM <tabelas>
    WHERE <valores> IN (SELECT <atribs>
    FROM <tabelas>
    WHERE <condição>);
  - Os valores de comparação em WHERE na consulta externa devem ser compatíveis com os atributos do SELECT da subconsulta

Q29 (subquery)

Selecionar o CPF de todos os empregados que trabalham no mesmo projeto e com a mesma quantidade de horas que o empregado cujo CPF é 012.345.678-90.

```
SELECT      DISTINCT cpfe
FROM        trabalha_em
WHERE       (numproj, horas)
            IN
            (SELECT numproj, horas
            FROM        trabalha_em
            WHERE       cpfe = 01234567890)
```

Q17 (conjuntos explícitos)

Recuperar os CPF de todos os empregados que trabalham nos projetos 1, 2 ou 3 (outra aplicação do operador IN).

```
SELECT      DISTINCT cpfe
FROM        trabalha_em
WHERE       numproj IN (1, 2, 3)
```

Q30 (subquery)

Selecionar o nome de todos os empregados cujo salário é maior do que o de todos os empregados do departamento 5.

```
SELECT      nome
FROM        empregado
WHERE       salario > ALL
            (SELECT    salario
            FROM       empregado
            WHERE      numdep = 5)
```

Q16A (subquery aninhada correlacionada)

Sempre que uma condição do WHERE de uma subconsulta faz referência a algum atributo de uma relação declarada na consulta externa, diz-se que as consultas são *correlacionadas*.

Recuperar o nome de todos os empregados que tenham um dependente homônimo e do mesmo sexo (!).

```
SELECT      e.nome
FROM        empregado e
WHERE       e.cpf IN
            (SELECT    cpfe
            FROM       dependente d
            WHERE      e.nome = d.nomedep
            AND        e.sexo = d.sexo)
```

SELECT

- Outros operadores de comparação para subconsultas:
  - = ANY, = SOME → retorna TRUE se o valor de comparação for igual a *algum* valor da lista (ou conjunto, resultante de um SELECT) que se segue; equivalentes a IN
  - Em vez de igual, pode-se usar <, >, >=, <=, <>
  - > ALL → retorna TRUE se o valor de comparação do que *todos* os valores da lista ou conjunto que se segue
- Pode-se ter diversos níveis de subconsultas
- Quando há ambigüidade entre nomes de atributos, o não qualificado pelo nome da tabela se refere à *subconsulta interna*

Q16B (EXISTS e UNIQUE)

EXISTS: verifica se a subconsulta produz algum resultado (Há também o NOT EXISTS)

UNIQUE: retorna TRUE se nãohouver nenhuma tupla repetida no resultado

Recuperar o nome de todos os empregados que tenham um dependente homônimo e do mesmo sexo (!).

```
SELECT      e.nome
FROM        empregado e
WHERE       EXISTS    (SELECT *
                       FROM dependente d
                       WHERE e.cpf = d.cpfe
                       AND   e.sexo = d.sexo
                       AND   e.nome = d.nomedep)
```

Q7 (EXISTS)

Relacionar os nomes dos gerentes que possuam pelo menos um dependente.

```
SELECT      e.nome
FROM        empregado e
WHERE       EXISTS    (SELECT *
                        FROM dependente d
                        WHERE e.cpf = d.cpfe
                        AND   EXISTS (SELECT *
                                      FROM departamento
                                      WHERE e.cpf=cpfger)
```

Q1A (tabelas de junção)

Recuperar o nome e o endereço de todos os empregados que trabalham no departamento cujo nome é ‘Pesquisa’.

```
SELECT    nome, endereco
FROM      (empregado e JOIN departamento d
          ON e.numdep = d.numdep)
WHERE     nomedep = ‘Pesquisa’
```

Como os nomes dos atributos na cláusula de junção são iguais, pode-se usar o NATURAL JOIN

```
SELECT    nome, endereco
FROM      (empregado NATURAL JOIN departamento)
WHERE     nomedep = ‘Pesquisa’
```

Q8B (Tabelas de junção)

Para cada empregado, recuperar o nome e o nome do supervisor responsável por ele.

Esta alternativa lista inclusive os nomes dos empregados que não têm supervisor, e portanto o s.nome fica nulo

```
SELECT    e.nome AS NOME_EMP,
          s.nome AS NOME_SUPER
FROM      (empregado e
          LEFT OUTER JOIN
          empregado s
          ON e.cpfsuper = s.cpf)
```

Tabelas de junção

- Em princípio, não há diferença de desempenho na execução de consultas expressas com tabelas de junção ou com a condição de junção na cláusula WHERE
- INNER JOIN = JOIN
  - INNER é default
- LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN = LEFT JOIN, RIGHT JOIN, FULL JOIN
  - OUTER é default
- NATURAL pode preceder qualquer tipo de JOIN
- ON só é necessário se não houver um par de atributos com o mesmo nome nas duas tabelas

Q19/Q20 (Agregações)

Encontrar a soma dos salários, o maior salário, o menor salário e a média de salários de todos os empregados.

```
SELECT    SUM(salario), MAX(salario),
          MIN(salario), AVG(salario)
FROM      empregado
```

Encontrar a soma dos salários, o maior salário, o menor salário e a média de salários de todos os empregados do departamento 5.

```
SELECT    SUM(salario), MAX(salario),
          MIN(salario), AVG(salario)
FROM      empregado
WHERE     numdep = 5
```

Q21/22 (Agregações)

Recuperar o número total de empregados da empresa.

```
SELECT    COUNT(*)
FROM      empregado
```

Recuperar o número de empregados do departamento ‘Pesquisa’

```
SELECT    COUNT(*)
FROM      empregado e, departamento d
WHERE     e.numdep = d.numdep
AND       d.nomedep = ‘Pesquisa’
```

Q23/Q5 (Agregações)

Contar o número de diferentes valores de salário na empresa.

```
SELECT    COUNT(DISTINCT salario)
FROM      empregado
```

Recuperar o nome de todos os empregados que têm mais de 2 dependentes

```
SELECT    nome
FROM      empregado e
WHERE     (SELECT COUNT(*)
          FROM dependente d
          WHERE e.cpf = d.cpfe) >= 2
```

Q24/Q25 (Agrupamento)

Para cada departamento, recuperar o seu número, número de empregados que nele trabalham e a média salarial.

```
SELECT    numdep, COUNT(*), AVG(salario)
FROM      empregado
GROUP BY  numdep
```

Recuperar o número, nome e número de empregados que trabalham em cada projeto

```
SELECT    numproj, nomeproj, COUNT(*)
FROM      projeto p, trabalha_em t
WHERE     p.numproj = t.numproj
GROUP BY  p.numproj, p.nomeproj
```

(a)

PNOME	MINICIAL	LNOME	SSN	• • •	SALARIO	SUPERSSN	DNO
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	• • •	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	null	1

Agrupamento das tuplas EMPREGADO por meio do valor de DNO

DNO	COUNT (*)	AVG (SALARIO)
5	4	33250
4	3	31000
1	1	55000

Resultado da Q24

(b)

PNOME	PNUMERO		ESSN	PNO	HORAS
ProdutoX	1		123456789	1	32,5
ProdutoX	1		453453453	1	20,0
ProdutoY	2		123456789	2	7,5
ProdutoY	2		453453453	2	20,0
ProdutoY	2		333445555	2	10,0
ProdutoZ	3		666884444	3	40,0
ProdutoZ	3		333445555	3	10,0
Automacao	10	• • •	333445555	10	10,0
Automacao	10		999887777	10	10,0
Automacao	10		987987987	10	35,0
Reorganizacao	20		333445555	20	10,0
Reorganizacao	20		987654321	20	15,0
Reorganizacao	20		888665555	20	null
NovosBeneficios	30		987987987	30	5,0
NovosBeneficios	30		987654321	30	20,0
NovosBeneficios	30		999887777	30	30,0

Esses grupos não são selecionados por HAVING, condição da Q26

Depois da aplicação da cláusula WHERE, mas antes da aplicação da cláusula HAVING

PNOME	PNUMERO		ESSN	PNO	HORAS
ProdutoY	2		123456789	2	7,5
ProdutoY	2		453453453	2	20,0
ProdutoY	2		333445555	2	10,0
Automacao	10	• • •	333445555	10	10,0
Automacao	10		999887777	10	10,0
Automacao	10		987987987	10	35,0
Reoganizacao	20		333445555	20	10,0
Reoganizacao	20		987654321	20	15,0
Reoganizacao	20		888665555	20	null
NovosBeneficios	30		987987987	30	5,0
NovosBeneficios	30		987654321	30	20,0
NovosBeneficios	30		999887777	30	30,0

Depois da aplicação da condição da cláusula HAVING

PNOME	COUNT (*)
ProdutoY	3
Automacao	3
Reorganizacao	3
NovosBenefícios	3

Resultado da Q26
(PNUMERO não apresentado)

Q26 (Agrupamento)

Para cada projeto em que trabalhem mais de dois empregados, recuperar o número do projeto, o nome e o número de empregados que nele trabalham.

```
SELECT    numproj, nomeproj, COUNT(*)
FROM      projeto p, trabalha_em t
WHERE     p.numproj = t.numproj
GROUP BY  numproj, nomeproj
HAVING    COUNT(*) > 2
```

HAVING incide sobre o resultado do agrupamento apenas.

Q28E (Agrupamento)

Para cada departamento, recuperar o número de empregados que nele trabalham e que ganham mais de R$5000, considerando apenas os departamentos que têm mais de 5 empregados.

```
SELECT    d.numdep, COUNT(*)
FROM      departamento d, empregado e
WHERE     d.numdep = e.numdep
AND       salario > 5000
GROUP BY  d.numdep
HAVING    COUNT(*) > 5
```

ERRADO: serão listados os departamentos em que mais de 5 empregados ganham mais de R$5000

Q28E (Agrupamento)

Versão CORRETA: Para cada departamento, recuperar o número de empregados que nele trabalham e que ganham mais de R$5000, considerando apenas os departamentos que têm mais de 5 empregados.

```
SELECT    d.numdep, COUNT(*)
FROM      departamento d, empregado e
WHERE     d.numdep = e.numdep
AND       salario > 5000
AND       numdep IN (SELECT numdep
                     FROM empregado
                     GROUP BY numdep
                     HAVING COUNT(*) > 5)
GROUP BY numdep
```

SELECT: visão geral

```
SELECT    <lista de atributos ou funções>
FROM      <lista de tabelas>
[WHERE    <condição>]
[GROUP BY <atributos de agrupamento>]
[HAVING   <condições sobre o agrupamento>]
[ORDER BY <lista de atributos>]
```

[ ] → opcional

Mais Comandos DML

- INSERT
  - INSERT INTO <tabela>
    VALUES (lista);
  - INSERT INTO <tabela>(atribs)
    VALUES (lista);
  - INSERT INTO <tabela>(atribs)
    SELECT... FROM... WHERE
  - Exemplos: A1, A2, A3 pg 228-229

Mais Comandos DML

- DELETE
  - DELETE FROM <tabela>
    WHERE <condição>;
  - DELETE FROM <tabela>
    WHERE <atrib> IN (SELECT... FROM ... WHERE)
  - DELETE FROM <tabela>

Mais Comandos DML

- UPDATE
  - UPDATE <tabela>
    SET <atributo>=<valor> [, <atributo>=<valor>]
    WHERE <condição>;
  - UPDATE <tabela>
    SET <atributo>=<valor> [, <atributo>=<valor>]
    WHERE <atributo> IN (SELECT... FROM... WHERE...);

Visões em SQL

- *Visão* é uma tabela que é derivada de outras tabelas
  - Não confundir com as visões do usuário, que foram discutidas na apresentação dos níveis de modelagem, cap. 1
- As tabelas das quais a visão é derivada podem ser *tabelas de base* (armazenadas no banco) ou outras visões
- Uma visão não necessariamente existe fisicamente; ela pode ser considerada uma *tabela virtual*
  - Isso pode limitar as operações de atualização via visões, mas não limita as consultas
- É um modo de deixar disponível uma consulta à qual é necessário fazer referência constantemente

Visões em SQL

- CREATE VIEW
  - CREATE VIEW <nome>
    AS SELECT... FROM... WHERE...
  - CREATE VIEW <nome>
    AS SELECT... FROM... WHERE... GROUP BY... HAVING...
- Acesso à visão: com SELECT, como em uma tabela comum
- DROP VIEW <nome>
- UPDATE: pode ser ou não possível. Vide seção 8.5.3, pg 232

Restrições Gerais como Assertivas

- Restrições mais gerais, definidas pelo usuário, e que não se enquadrem nas possibilidades do CREATE TABLE podem ser implementadas usando *assertivas declarativas*
- Comando CREATE ASSERTION
  - CREATE ASSERTION <nome>
    CHECK (<condição>);
- A condição é especificada como uma consulta

Referências e Leitura complementar

- Elmasri & Navathe, cap. 8: SQL-99
- O capítulo 9 inclui técnicas de programação usando SQL
  - Embutido em linguagens hospedeiras, usando cursores ou recuperando tuplas isoladas
  - Especificando consultas em tempo de execução usando SQL dinâmico
  - Por meio de chamada de funções (JDBC)
  - Stored procedures