Restrições, Selects Simples e Inner Joins

Comentar sobre o tipo BINARY LARGE OBJECT (BLOB) que pode ser usado para armazenar grandes valores binários como imagens.

CONSTRAINTS e DEFAULTS:

- NOT NULL: Define que um atributo não pode receber valor NULL.
- DEFAULT <valor>: utilizada para especificar valores padrão para as colunas quando os valores não forem dados no comando INSERT. Exemplo: Cpf_gerente CHAR(11) NOT NULL **DEFAULT '88866555576'**;
- Outro tipo de restrição pode limitar valores de atributo ou domínio usando a cláusula **CHECK** (verificação) após uma definição de atributo ou domínio. Por exemplo, suponha que números de departamento sejam restritos a números inteiros entre 1 e 20; então, declarar o atributo de Dnumero como: Dnumero INT NOT NULL **CHECK (Dnumero > 0 AND Dnumero < 21)**;
- **PRIMARY KEY**: definem campos usados como identificadores únicos das linhas. É utilizada para construir índices de acesso rápido aos dados da tabela. Se uma chave primária tiver um único atributo, a cláusula pode acompanhar o atributo diretamente, por exemplo, Dnumero INT **PRIMARY KEY**; . Senão, pode ser usada a sintaxe a seguir:

```
CREATE TABLE employees
(
  employee_id INTEGER,
  last_name VARCHAR NOT NULL,
  first_name VARCHAR,
  hire_date DATE,
  CONSTRAINT employees_pk PRIMARY KEY (employee_id)
);
```

A expressão CONSTRAINT <nome> PRIMARY KEY (<coluna 1>, <coluna 2>, ...) é usada para criar uma chave primária composta pelas colunas mencionadas e atribuir o nome especificado para a restrição. O nome pode ser usado a posteriori para destruir a restrição.

Importante: No sqlite3, ao definir que um atributo inteiro é um campo PRIMARY KEY, se o valor não for especificado ao inserir uma nova linha, ele receberá um valor inteiro único.

- **UNIQUE**: especifica que os valores de um atributo devem ser únicos. Serve para indiar chaves alternativas (secundárias). A cláusula UNIQUE também pode ser especificada diretamente para uma chave secundária se esta for um único atributo, como no exemplo a seguir: email VARCHAR(32) **UNIQUE**; .
- FOREIGN KEY: é utilizada para definir restrições de **integridade referencial**.
    - A restrição de integridade referencial é especificada entre duas relações e usada para manter a consistência entre tuplas nas duas relações. Informalmente, a restrição de integridade referencial afirma que uma tupla em uma relação que

referencia outra relação precisa se referir a uma tupla existente nessa relação. Por exemplo, o atributo n_depto de FUNCIONARIO fornece o código de departamento para o qual cada funcionário trabalha; logo, seu valor em cada tupla FUNCIONARIO precisa combinar com o valor de depto_codigo de alguma tupla na relação DEPARTAMENTO.

- Uma restrição de integridade referencial pode ser violada quando tuplas são inseridas ou excluídas, ou quando um valor de atributo de chave estrangeira ou chave primária é modificado. A ação default que a SQL toma para uma violação de integridade é rejeitar a operação de atualização que causará uma violação. O projetista do esquema pode especificar uma ação alternativa:
  - Uma opção deve ser qualificada com ON DELETE ou ON UPDATE.
  - Opções: SET NULL, CASCADE, SET DEFAULT.
  - **Importante**: A restrição ON DELETE CASCADE é usada para excluir as linhas da tabela filha automaticamente, quando as linhas da tabela pai são excluídas. Por exemplo, quando um aluno se registra em uma plataforma de aprendizado on-line, todos os detalhes do aluno são registrados com seu número/id exclusivo. Um aluno pode se matricular em um ou mais cursos. Suponha que você exclua uma linha da tabela "Aluno", o ON DELETE CASCADE pode ser usado para excluir todas as linhas da tabela "Matrícula" que fazem referência à linha da tabela "Aluno".
  - A ação para CASCADE ON UPDATE é mudar o valor do(s) atributo(s) de chave estrangeira de referência para o (novo) valor de chave primária atualizado para todas as tuplas de referência.

Exemplo: Como resultado das restrições de chave estrangeira utilizadas na tabela Enroll, a matrícula será deletada sempre que o curso ou o estudante sejam deletados.

```
CREATE TABLE Student (
   sno INT PRIMARY KEY,
   sname VARCHAR(20),
   age INT
);

CREATE TABLE Course (
   cno INT PRIMARY KEY,
   cname VARCHAR(20)
);
```

```
CREATE TABLE Enroll (
   sno INT,
   cno INT,
   jdate date,
   PRIMARY KEY(sno, cno),
   FOREIGN KEY(sno)
      REFERENCES Student(sno)
      ON DELETE CASCADE,
   FOREIGN KEY(cno)
      REFERENCES Course(cno)
      ON DELETE CASCADE
);
```

```
CREATE TABLE FUNCIONARIO
    ( . . . ,
    Dnr             INT             NOT NULL      DEFAULT 1,
    CONSTRAINT CHPFUNC
        PRIMARY KEY (Cpf),
    CONSTRAINT CHESUPERFUNC
        FOREIGN KEY (Cpf_supervisor) REFERENCES FUNCIONARIO(Cpf)
        ON DELETE SET NULL    ON UPDATE CASCADE,
    CONSTRAINT CHEDEPFUNC
    FOREIGN KEY(Dnr) REFERENCES DEPARTAMENTO(Dnumero)
            ON DELETE SET DEFAULT ON UPDATE CASCADE);
CREATE TABLE DEPARTAMENTO
    ( . . . ,
    Cpf_gerente     CHAR(11)      NOT NULL      DEFAULT '88866555576',
    . . . ,
    CONSTRAINT CHPDEP
        PRIMARY KEY(Dnumero),
    CONSTRAINT CHSDEP
        UNIQUE (Dnome),
    CONSTRAINT CHEGERDEP
    FOREIGN KEY (Cpf_gerente) REFERENCES FUNCIONARIO(Cpf)
        ON DELETE SET DEFAULT ON UPDATE CASCADE);
CREATE TABLE LOCALIZACAO_DEP
    ( . . . ,
    PRIMARY KEY (Dnumero, Dlocal),
    FOREIGN KEY (Dnumero) REFERENCES DEPARTAMENTO(Dnumero)
        ON DELETE CASCADE    ON UPDATE CASCADE);
```

Aqui, o projetista de banco de dados escolhe ON DELETE SET NULL e ON UPDATE CASCADE para a chave estrangeira Cpf_supervisor de FUNCIONARIO. Isso significa que, se a tupla para um funcionário supervisor é excluí- da, o valor de Cpf_supervisor será automaticamente definido como NULL para todas as tuplas de funcionários que estavam referenciando a tupla do funcionário excluído. Por sua vez, se o valor de Cpf para um funcionário supervisor é atualizado (digamos, porque foi inserido incorretamente), o novo valor será propagado em cascata de Cpf_supervisor para todas as tuplas de funcionário que referencia a tupla de funcionário atualizada.

Importante: Algumas chaves estrangeiras que podem causar erros, pois são especificadas por referências circulares ou porque dizem respeito a uma tabela que ainda não foi criada. Por exemplo, a chave estrangeira Cpf_supervisor na tabela FUNCIONARIO é uma referência circular, pois se refere à própria tabela. A chave estrangeira Dnr na tabela FUNCIONARIO se refere à tabela DEPARTAMENTO, que ainda não foi criada. Para lidar com esse tipo de problema, essas restrições podem ser omitidas inicialmente do comando CREATE TABLE, e depois acrescentadas usando a instrução ALTER TABLE.

Exemplo: Implementar todas as restrições na biblioteca.

Banco de Dados usado nas seções que explicam comandos de seleção

O banco de dados contém dados de uma empresa.

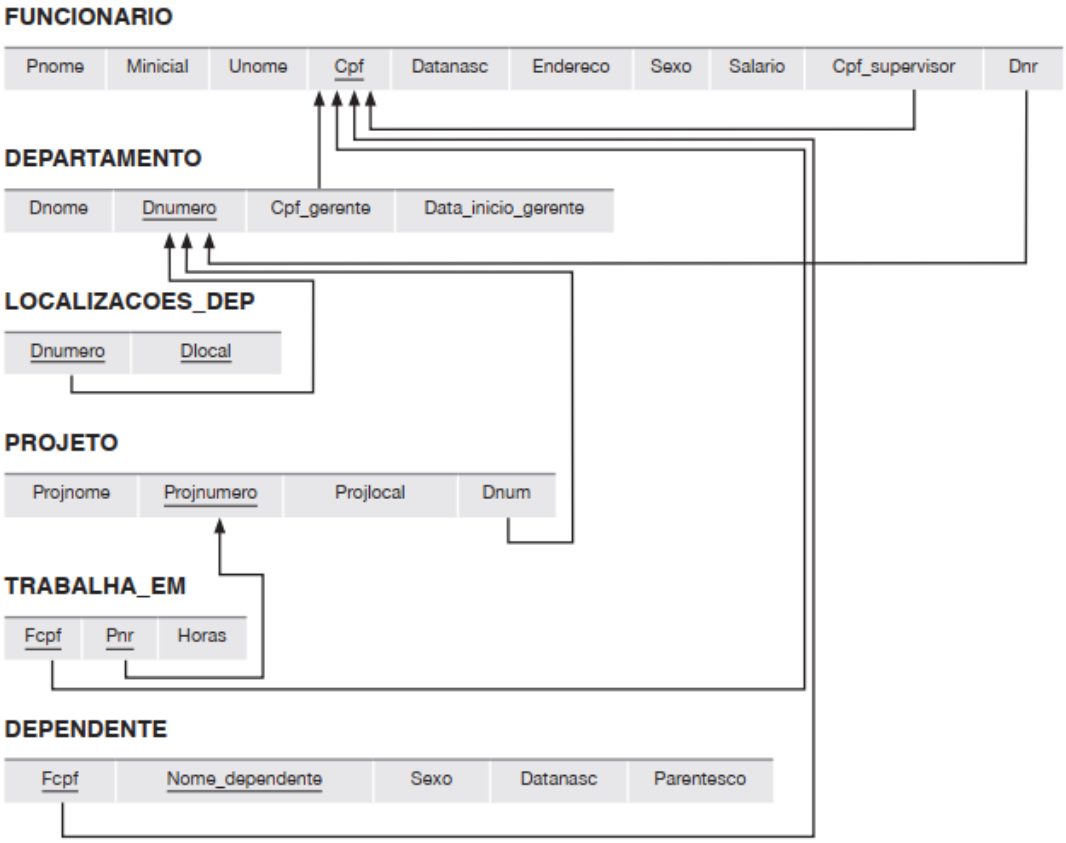

Figura 3.7
Restrições de integridade referencial exibidas no esquema de banco de dados relacional EMPRESA.

SELECÕES SIMPLES E INNER JOINS

As seleções mais simples possuem a sintaxe:

```
SELECT <lista atributos> FROM <lista tabelas> WHERE <condição>;
```

A lista de atributos, chamados atributos de projeção, permite selecionar um subconjunto dos atributos. As informações de duas ou mais tabelas podem ser unidas usando a operação de INNER JOIN. O exemplo abaixo ilustra como recuperar as informações de livros que ainda não foram devolvidos junto com as informações do locatário e do livro.

```
SELECT *
    FROM livro
    INNER JOIN emprestimo ON (livro.codigo = emprestimo.livro)
    INNER JOIN usuario ON (usuario.cpf = emprestimo.cpf)
     WHERE emprestimo.entrega is NULL;
```

O mesmo comando poderia ser escrito escrevendo as tabelas no FROM e as condições de junção na cláusula WHERE:

```
SELECT *
    FROM livro, emprestimo, usuario
    WHERE livro.codigo = emprestimo.livro AND usuario.cpf = emprestimo.cpf AND emprestimo.entrega is NULL;
```

OUTROS OPERADORES DE SELEÇÃO

- Se quisermos eliminar tuplas duplicadas do resultado de uma consulta SQL, usamos a palavra-chave **DISTINCT** na cláusula **SELECT**, significando que apenas as tuplas distintas deverão permanecer no resultado.

```
SELECT DISTINCT salario FROM funcionario;
```

- comparação apenas sobre partes de uma cadeia de caracteres, usando o operador de comparação **LIKE**. Isso pode ser usado para combinação de padrão de cadeia. Cadeias parciais são especificadas usando dois caracteres reservados:
    - % substitui um número qualquer de zero ou mais caracteres.
    - o sublinhado (_) substitui um único caractere.

```
Recuperar todos os funcionários cujo endereço esteja em São Paulo, SP
SELECT Pnome, Unome FROM FUNCIONARIO WHERE Endereco LIKE '%SaoPaulo,SP%';

Encontrar todos os funcionários que nasceram durante a década de 1950.
SELECT Pnome, Unome FROM FUNCIONARIO WHERE Datanasc LIKE '_ _ _ _ _ _ _ _ 5 _';
```

- Outro recurso permite o uso de aritmética nas consultas. Os operadores aritméticos padrão para adição (+), subtração (–), multiplicação (*) e divisão (/) podem ser aplicados a valores ou atributos numéricos com domínios numéricos.

```
Mostrar os salários resultantes se cada funcionário que trabalha no projeto 'ProdutoX' receber um aumento de 10 por cento.

SELECT F.Pnome, F.Unome, 1,1 * F.Salario AS Aumento_salario FROM FUNCIONARIO AS F, TRABALHA_EM AS T, PROJETO AS P WHERE F.Cpf=T.Fcpf AND T.Pnr=P.Projnumero AND P.Projnome='ProdutoX';
```

- Outro operador de comparação, que pode ser usado por conveniência, é BETWEEN:

```
Recuperar todos os funcionários no departamento 5 cujo salário esteja entre R$ 30.000 e R$ 40.000.

SELECT * FROM FUNCIONARIO WHERE (Salario BETWEEN 30.000 AND 40.000) AND Dnr = 5;
```

Ordenação

- SQL permite que o usuário ordene as tuplas no resultado de uma consulta pelos valores de um ou mais dos atributos que aparecem, usando a cláusula ORDER BY.

Recuperar uma lista dos funcionários e dos projetos em que estão trabalhando, ordenada por departamento e, dentro de cada departamento, ordenada alfabeticamente pelo sobrenome, depois pelo nome.

```
SELECT D.Dnome, F.Unome, F.Pnome, P.Projnome FROM DEPARTAMENTO D, FUNCIONARIO F, TRABALHA_EM T, PROJETO P WHERE D.Dnumero= F.Dnr AND F.Cpf= T.Fcpf AND T.Pnr= P.Projnumero ORDER BYD.Dnome, F.Unome, F.Pnome;
```

Ordem Crescente ou Decrescente

Podemos especificar a palavra-chave DESC se quisermos ver o resultado em uma ordem decrescente de valores. A palavra-chave ASC pode ser usada para especificar a ordem crescente explicitamente.

```
(...) ORDER BY D.Dnome DESC, F.Unome ASC, F.Pnome ASC
```

Funções de Agregação

As funções de agregação são usadas para resumir informações de várias tuplas em uma síntese de tupla única. O agrupamento é usado para criar subgrupos de tuplas antes do resumo.

Existem diversas funções de agregação embutidas: COUNT, SUM, MAX, MIN e AVG.

Achar a soma dos salários de todos os funcionários do departamento ‘Pesquisa’, bem como o salário máximo, o salário mínimo e a média dos salários nesse departamento.

```
SELECT SUM (Salario), MAX (Salario), MIN (Salario), AVG (Salario) FROM (FUNCIONARIO JOIN DEPARTAMENTO ON Dnr=Dnumero) WHERE Dnome=‘Pesquisa’;
```

GROUP BY

Em muitos casos, queremos aplicar as funções de agregação a subgrupos de tuplas em uma relação, na qual os subgrupos são baseados em alguns valores de atributo. Por exemplo, podemos querer achar o salário médio dos funcionários em cada departamento ou o número de funcionários que trabalham em cada projeto. Nesses casos, precisamos particionar a relação em subconjuntos de tuplas (ou grupos) não sobrepostos. Cada grupo (partição) consistirá nas tuplas que possuem o mesmo valor de algum(ns) atributo(s), chamado(s) atributo(s) de agrupamento. Podemos, então, aplicar a função a cada grupo desse tipo independentemente, para produzir informa- ções de resumo sobre cada grupo. A SQL tem uma cláusula GROUP BY para essa finalidade. A cláusula GROUP BY especifica os atributos de agrupamento, que também devem aparecer na cláusula SELECT.

Consulta 24. Para cada departamento, recuperar o número do departamento, o número de funcionários no departamento e seu salário médio.

```
C24: SELECT     Dnr, COUNT (*), AVG (Salario)
     FROM       FUNCIONARIO
     GROUP BY   Dnr;
```

Na consulta, as tuplas FUNCIONARIO são divididas em grupos — cada grupo tendo o mesmo valor para o atributo de agrupamento Dnr. Logo, cada grupo contém os funcionários que trabalham no mesmo departamento. As funções COUNT e AVG são aplicadas a cada grupo de tuplas.

HAVING

Às vezes, queremos recuperar os valores dessas funções somente para grupos que satisfazem certas condições. Por exemplo, suponha que queremos modificar a Consulta 25 de modo que apenas projetos com mais de dois funcionários apareçam no resultado. A SQL oferece uma cláusula HAVING, que pode aparecer em conjunto com uma cláusula GROUP BY, para essa finalidade. A cláusula HAVING oferece uma condição sobre a informação de resumo referente ao grupo de tuplas associado a cada valor dos atributos de agrupamento. Somente os grupos que satisfazem a condição são recuperados no resultado da consulta.

Consulta 26. Para cada projeto *em que mais de dois funcionários trabalham*, recupere o número e o nome do projeto e o número de funcionários que trabalham no projeto.

```
C26: SELECT     Projnumero, Projnome, COUNT (*)
     FROM       PROJETO, TRABALHA_EM
     WHERE      Projnumero=Pnr
     GROUP BY   Projnumero, Projnome
     HAVING     COUNT (*) > 2;
```

Observe que precisamos ter um cuidado extra quando duas condições diferentes se aplicam (uma para a função de agregação na cláusula SELECT e outra para a função na cláusula HAVING). Por exemplo, suponha que queremos contar o número total de funcionários cujos salários são superiores a R$ 40.000 em cada departamento, mas somente para os departamentos em que há mais de cinco funcionários trabalhando.

Incorreta

```
SELECT     Dnome, COUNT (*)
FROM       DEPARTAMENTO, FUNCIONARIO
WHERE      Dnumero=Dnr AND Salario>40.000
GROUP BY   Dnome
HAVING     COUNT (*) > 5;
```

Correta

```
C28: SELECT Dnumero, COUNT (*)
     FROM   DEPARTAMENTO, FUNCIONARIO
     WHERE  Dnumero=Dnr AND Salario>40.000 AND
            ( SELECT  Dnr IN
              FROM  FUNCIONARIO
              GROUP BY Dnr
              HAVING  COUNT (*) > 5)
```

A consulta da esquerda está incorreta porque selecionará somente departamentos que tenham mais de cinco funcionários que ganham cada um mais de R$ 40.000. A regra é que a cláusula WHERE é executada primeiro, para selecionar as tuplas individuais ou tuplas de junção; a cláusula HAVING é aplicada depois, para selecionar grupos individuais de tuplas. Logo, as tuplas já estão restritas a funcionários que ganham mais de R$ 40.000 antes que a cláusula HAVING seja aplicada.

RESUMO

SELECT <lista atributo e função>
FROM <lista tabela>
[**JOINS**]
[**WHERE** <condição>]
[**GROUP BY** <atributo(s) de agrupamento>]
[**HAVING** <condição de grupo>]
[**ORDER BY** <lista atributos>];