Resumo 3

Marcos Geraldo Braga Emiliano 19.1.4012

Banco de Dados 1

Fonte: Slides das aulas

-Álgebra Relacional

Um modelo de dados, além de definir estruturas e restrições, deve definir um conjunto de operações para manipular os dados. A álgebra relacional constitui o conjunto básico de operações do modelo relacional, as operações possibilitam especificar solicitações básicas de recuperação, sendo que o resultado de uma recuperação é uma nova relação, formada a partir de uma ou mais relações.

As operações de recuperação se dividem em: operações específicas de bancos de dados relacionais: seleção, projeção, junção, entre outras, ou operações da teoria de conjuntos: união, interseção, diferença e produto cartesiano.

Empregado

PrimeiroNome	InicialMeio	UltimoNome	NumEmpregado	DataNascimento	Endereco	Sexo	Salario	NumSupervisor	NumDepto
João	B	Silva	123456789	09/01/65	R. da Bahia, 2557	M	300.00	333445555	5
Frank	T	Santos	333445555	08/12/55	Av. Afonso Pena, 3005	M	4000.00	888665555	5
Alice	J	Pereira	999887777	19/07/68	Av. do Contorno, 2534	F	2500.00	987654321	4
Luciene	S	Ferreira	987654321	20/06/51	R. Iraí, 175	F	430.00	888665555	4
Pedro	K	Magalhães	666884444	15/09/52	Av. Silva Lobo, 2050	M	1200.00	333445555	5
Daniela	A	Oliveira	453453453	31/07/62	R. Ataliba Lago, 250	F	2500.00	333445555	5
Mateus	V	Mascarenhas	987987987	29/03/79	R. Contria, 12	M	2500.00	987654321	4
Fábio	E	Lemos	888665555	10/11/47	R. Chile, 425	M	5500.00	null	1

Departamento

NomeDepto	NumDepto	NumGerente	DataInicioGerencia
Pesquisa	5	333445555	22/05/98
Administração	4	987654321	01/01/95
Diretoria	1	888665555	19/06/01

Localizacao_Depto

NumDepart	Localizacao
1	Savassi
4	Centro
5	Buritis
5	Pampulha
5	Contagem

Projeto

NomeProj	NumProj	Localizacao	NumDepto
Produto X	1	Buritis	5
Produto Y	2	Pampulha	5
Produto Z	3	Contagem	5
Informatização	10	Centro	4
Reorganização	20	Savassi	1
NovosBenefícios	30	Centro	4

Trabalha_em

NumEmpregado	NumProj	Horas
123456789	1	32
123456789	2	7
666884444	3	40
453453453	1	20
453453453	2	20
333445555	2	10
333445555	3	10
333445555	10	10
333445555	20	10
999887777	30	30
999887777	10	10
987987987	10	35
987987987	30	5
987654321	30	20
987654321	20	15
888665555	20	null

Dependente

NumEmpregado	NomeDependente	Sexo	DataAniversario	Parentesco
333445555	Aline	F	03/04/76	Filha
333445555	Vitor	M	25/10/73	Filho
333445555	Joana	F	03/05/98	Cônjuge
987654321	Igor	M	29/02/52	Cônjuge
123456789	Michel	M	01/01/88	Filho
123456789	Aline	F	31/12/98	Filha
123456789	Elizabeth	F	05/05/57	Cônjuge

-Operação Seleção

A operação Seleção é utilizada para selecionar um conjunto de tuplas de uma relação: $\sigma_{<cond>}(<R>)$, onde <cond> é uma condição de seleção e <R> é o nome de uma relação. Ex.: selecionar todos os empregados que trabalham no departamento 5. $\sigma_{NumDepto=5}$ (Empregado).

É uma operação unária (feita em uma única relação), o grau (número de atributos) da relação resultante é o mesmo da relação original, a operação é comutativa: $\sigma_{<cond1>}(\sigma_{<cond1>}(<R>)) = \sigma_{<cond1>}(\sigma_{<cond1>}(<R>))$

Pode-se combinar uma cascata de operações Seleção em uma única operação Seleção: $\sigma_{<cond1>}(\sigma_{<cond2>}(\sigma_{<cond3>}(<R>))) = \sigma_{<cond1> \text{ E } <cond2> \text{ E } <cond3>}(<R>)$

-Operação Projeção

A operação Projeção é utilizada para selecionar um conjunto de atributos de uma relação: $\pi_{<atributos>}(<R>)$, onde <atributos> é uma lista de atributos dentre os atributos da relação R e <R> é o nome de uma relação. Ex.: listar o nome e o salário de todos os empregados. $\pi_{\text{PrimeiroNome, UltimoNome, Salario}}(\text{Empregado})$

É uma operação unária (feita em uma única relação), caso a lista de atributos inclua somente atributos que não sejam chaves de R, é possível que ocorram tuplas duplicadas, a operação Projeção remove tuplas duplicadas de tal forma que o resultado seja uma relação válida, com isso, o número de tuplas na relação resultante é sempre menor ou igual ao número de tuplas da relação R, a operação não é comutativa; pode-se dizer que: $\pi_{<lista1>}(\pi_{<lista2>}(<R>)) = \pi_{<lista1>}(<R>)$

-Sequência de Operações

É comum aplicar diversas operações da álgebra relacional, uma após a outra (sequência de operações), pode-se escrever as operações na forma de uma única expressão ou aplicar uma operação a cada vez, criando relações de resultado intermediário; neste último caso, deve-se nomear as relações envolvidas. Ex.: listar o nome e o salário de todos os empregados que trabalham no departamento de número 5.

$\pi_{\text{PrimeiroNome, UltimoNome, Salario}}(\sigma_{\text{NumDepto}} = 5 (\text{Empregado}))$

ou

$\text{Dep5\_Emps} \leftarrow \sigma_{\text{NumDepto} = 5}(\text{Empregado})$

$\text{Resultado} \leftarrow \pi_{\text{PrimeiroNome, UltimoNome, Salario}}(\text{Dep5\_Emps})$

Pode-se utilizar a técnica "sequência de operações" para renomear os atributos nas relações intermediárias e de resultado: basta listar os nomes dos novos atributos entre parênteses juntamente com os nomes das novas relações:

$\text{Dep5\_Emps} \leftarrow \sigma_{\text{NumDepto} = 5}(\text{Empregado})$

$\text{Resultado (PNome, UNome, Sal)} \leftarrow \pi_{\text{PrimeiroNome, UltimoNome, Salario}}(\text{Dep5\_Emps})$

Caso nenhuma renomeação seja aplicada em uma Seleção, os nomes dos atributos na relação resultante são os mesmos da relação original e estarão na mesma ordem, para uma Projeção sem renomeação, a relação resultante possui os mesmos nomes dos atributos especificados na lista de projeção e aparecem na mesma ordem da lista.

-Operação Renomeação

A operação Renomeação é utilizada para renomear uma relação ou atributos da mesma: $\rho_{S(b1,b2,...,bn)}(<R>)$ ou $\rho S(<R>)$ ou $\rho_{(b1,b2,...,bn)}(<R>)$, onde <S> é o novo nome da relação, <b1, b2, ..., bn> são os novos nomes dos atributos e <R> é a relação original. A primeira expressão renomeia tanto a relação quanto os atributos, a segunda renomeia apenas a relação e a terceira renomeia apenas os atributos. Ex.: listar o nome e o salário de todos os empregados que trabalham no departamento de número 5.

$Dep5\_Emps \leftarrow \sigma_{NumDepto=5}$ (Empregado)

$\rho_{Resultado}$ (PNome, UNome, Sal)

($\pi_{PrimeiroNome,UltimoNome,Salario}$ (Dep5_Emps))

-Operações Teóricas de Conjuntos

A álgebra relacional possui um grupo padrão de operações matemáticas sobre conjuntos, as operações são binárias, ou seja, envolvem duas relações, para algumas operações, as relações devem possuir o mesmo tipo de tuplas, sendo consideradas compatíveis para união. Duas relações R(A1,A2,...,An) e S(B1,B2,...,Bn) são compatíveis para união se possuírem o mesmo grau "n" e se dom(Ai) = dom(Bi) para 1 <= i <= n.

As operações teóricas de conjuntos que exigem relações compatíveis para união são: União, denotada por $R \cup S$, gera uma relação que inclui todas as tuplas que estão em R ou em S ou em ambas, Interseção, denotada por $R \cap S$, gera uma relação que inclui todas as tuplas que estão tanto em R quanto em S, Diferença, denotada por R - S, gera uma relação que inclui todas as tuplas que estão em R, mas não estão em S. A relação resultante das operações possui os mesmos nomes de atributos da primeira relação (R) envolvida nas operações.

As operações de união e interseção são comutativas e associativas:

$R \cup S = S \cup R$ e $R \cap S = S \cap R$

$R \cup (S \cup T) = (R \cup S) \cup T$ e $(R \cap S) \cap T = R \cap (S \cap T)$

Ex.: listar o número de todos os empregados que trabalham no departamento 5 ou que supervisionam diretamente um empregado que trabalhe no departamento 5.

$Dep5\_Emps \leftarrow \sigma_{NumDepto=5}$ (Empregado)

$Result1 \leftarrow \pi_{NumEmpregado}$ (Dep5_Emps)

Result2 (NumEmpregado) $\leftarrow \pi_{NumSupervisor}$ (Dep5_Emps)

Resultado $\leftarrow$ Result1 $\cup$ Result2

A operação de conjunto binária Produto Cartesiano, representada por ×, é utilizada para combinar tuplas de duas relações de forma combinatória, as relações não precisam ser compatíveis para união. O resultado de $R(A_1,A_2,...,A_n)$ × $S(B_1,B_2,...,B_m)$ é uma relação $Q(A_1,A_2,...,A_n, B_1,B_2,...,B_m)$, com "n+m" atributos, a relação Q possui uma tupla para cada combinação de tuplas das relações envolvidas: se R possui n_R tuplas e S possui n_S tuplas, então Q possuirá $n_R * n_S$ tuplas. Não é uma operação muito usual pois gera tuplas que não fazem sentido, torna-se prática quando é seguida por uma Seleção que combina valores de atributos nas relações envolvidas. Ex.: listar, para cada empregado do sexo feminino, os nomes dos seus dependentes.

Emps_Mulheres ← $\sigma_{Sexo='F'}$ (Empregado)

Nomes_Emp (Nome, Sobrenome, NumEmp) ← $\pi_{PrimeiroNome, UltimoNome, NumEmpregado}$ (Emps_Mulheres)

Deps_Emp ← Nomes_Emp × Dependente

Deps_Certos ← $\sigma_{NumEmp = NumEmpregado}$ (Deps_Emp)

Resultado ← $\pi_{Nome, Sobrenome, NomeDependente}$ (Deps_Certos)

Uma vez que a operação Produto Cartesiano, seguida da operação Seleção, é utilizada com frequência, foi definida uma operação especial, denominada Junção, para especificar tal sequência como uma única operação.

-Operação Junção

A operação Junção é utilizada para combinar tuplas relacionadas de duas relações em uma única tupla: R **JUN** $_{<cond>}$ S, onde R e S são relações e <cond> é uma condição de junção entre as relações.

$R(A_1,A_2,...,A_n)$ **JUN** $_{<cond>}$ $S(B_1,B_2,...,B_m)$ gera uma relação $Q(A_1,A_2,...,A_n, B_1,B_2,...,B_m)$, com "n+m" atributos. $Q(A_1,A_2,...,A_n, B_1,B_2,...,B_m)$, com "n+m" atributos, a relação Q possui uma tupla para cada combinação de tuplas das relações envolvidas, sempre que a combinação satisfizer a condição de junção.

Uma condição geral de junção é: <cond1> e <cond2> e ... e <condN>, onde cada condição é da forma A_i θ B_j: A_i é atributo de R, B_j é atributo de S de mesmo domínio de A_i, e θ é um operador de comparação {=, <, ≤, >, ≥, ≠}. Ex.: listar, para cada empregado do sexo feminino, os nomes dos seus dependentes.

Emps_Mulheres ← $\sigma_{Sexo='F'}$ (Empregado)

Nomes_Emp (Nome, Sobrenome, NumEmp) ← $\pi_{PrimeiroNome, UltimoNome, NumEmpregado}$ (Emps_Mulheres)

Deps_Certos ← Nomes_Emp **JUN** $_{NumEmp = NumEmpregado}$ Dependente

Resultado ← $\pi_{Nome, Sobrenome, NomeDependente}$ (Deps_Certos)

Ex.: listar o nome do gerente de cada departamento.

Ger_Dep ← Departamento **JUN** $_{\text{NumGerente = NumEmpregado}}$ Empregado

Resultado ← $\pi_{\text{NomeDepto, PrimeiroNome, UltimoNome}}$ (Ger_Dep)

A operação Junção mais comum, denominada Equijunção, envolve apenas condições de junção com comparações de igualdade. Uma Equijunção onde os dois atributos da comparação têm o mesmo nome é chamada Junção Natural, sendo denotada por $*$; neste caso, apenas um dos atributos da comparação aparece na relação resultante e a condição de junção não é especificada.

Ex.: listar, para cada empregado do sexo feminino, os nomes dos seus dependentes.

Emps_Mulheres ← $\sigma_{\text{Sexo='F'}}$ (Empregado)

Deps_Certos ← Emps_Mulheres * Dependente

Resultado ← $\pi_{\text{PrimeiroNome, UltimoNome, NomeDependente}}$ (Deps_Certos)

-Operação Divisão

A operação binária Divisão, representada por ÷, é utilizada para um tipo especial de consulta que ocorre, algumas vezes, em aplicações de bancos de dados.

A operação Divisão R(Z) ÷ S(X) só pode ser aplicada se $X \subseteq Z$. O resultado da divisão é uma relação T contendo o conjunto de atributos de R que não são atributos de S, ou seja, os atributos Z-X. Uma tupla de T é formada por valores dos atributos Z-X de R cujos valores referentes dos atributos X de R combinaram com todos os valores dos atributos X de S.

Ex.: listar o nome dos empregados que trabalham em todos os projetos nos quais "João Silva" trabalha.

1º passo: recuperar os números de projetos nos quais "João Silva" trabalha.

Joao ← $\sigma_{\text{PrimeiroNome='João' E UltimoNome='Silva'}}$ (Empregado)

Projs_Joao ← π_{NumProj} (Trabalha_em $*$ Joao)

2º passo: filtrar os atributos desejados da relação Trabalha_em.

NumEmps_NumProjs ← $\pi_{\text{NumEmpregado, NumProj}}$ (Trabalha_em)

3º passo: aplicar a Divisão entre as duas relações geradas; o resultado conterá os números dos empregados desejados.
NumEmps ← NumEmps_NumProjs ÷ Projs_Joao

Resultado ← $\pi_{\text{PrimeiroNome, UltimoNome}}$ (NumEmps $*$ Empregado)

-Funções de Agregação e Agrupamento

Uma solicitação que pode ser expressa na álgebra relacional é a aplicação de funções matemáticas de agregação em coleção de valores do banco de dados, as funções mais comuns aplicadas a coleção de valores numéricos são: Sum (soma), Average (média), Maximum (máximo), Minimum (mínimo), Count (contador de tuplas).

Outra solicitação envolve o agrupamento de tuplas de uma relação por meio dos valores de alguns atributos e, logo após, a aplicação de uma função de agregação em cada grupo. Um exemplo é agrupar as tuplas de empregados pelo "NumDepto": cada grupo é composto pelas tuplas de empregados que trabalham em um mesmo departamento.

Uma função de agrupamento é definida da seguinte forma: <atributos de agrupamento> ℑ <funções de agregação> (<R>); onde <R> é uma relação, <atributos de agrupamento> é uma lista de atributos de R responsável pelo agrupamento e <funções de agregação> é uma lista de pares de (<função><atributo>): em cada um destes pares, <função> é uma das funções de agregação permitidas e <atributo> é um atributo de R cuja função de agregação será aplicada.

A relação resultante possui os atributos de agrupamento e os resultados gerados pelas funções de agregação, haverá uma tupla para cada grupo gerado pelos atributos de agrupamento. Ex.: listar o nº de cada departamento, a quantidade de empregados em cada um e a média de seus salários.

ρResultado (NumD, NumEmps, MediaSals) (NumDepto ℑ Count NumEmpregado, Average Salario (Empregado))

Caso não seja aplicada nenhuma renomeação, os atributos da relação resultante correspondentes às funções de agregação são, cada um deles, a concatenação do nome da função e o nome do atributo (<função>_<atributo>).
No exemplo NumDepto ℑ Count NumEmpregado, Average Salario (Empregado); os atributos da relação resultante são: NumDepto, Count_NumEmpregado, Average_Salario.

Caso não seja especificado algum atributo de agrupamento, as funções de agregação são aplicadas em todas as tuplas da relação envolvida, a relação resultante terá uma só tupla. O exemplo ℑ Count NumEmpregado, Average Salario (Empregado) recupera a quantidade total de empregados e a média de seus salários (atributos Count_NumEmpregado e Average_Salario).

Fechamento Recursivo
Um tipo de operação que pode ser expresso na álgebra relacional é o fechamento recursivo, tal operação é aplicada a um auto-relacionamento entre tuplas do mesmo tipo. Ex.: listar todos os empregados que são supervisionados, em todos os níveis, pelo empregado "Fábio Lemos".

No nível 1, tem-se:

Fabio_Num ← $\pi_{NumEmpregado}(\sigma_{PrimeiroNome='Fábio' \text{ E } UltimoNome='Lemos'}(Empregado))$

Supervisao (NumEmp, NumSup) ← $\pi_{NumEmpregado,\ NumSupervisor}$ (Empregado)

Resultado1(Num) ← π_{NumEmp}(Supervisao **JUN** $_{NumSup\ =\ NumEmpregado}$ Fabio_Num)

No nível 2, tem-se:

Resultado2 (Num) ← π_{NumEmp} (Supervisao **JUN** $_{NumSup\ =\ Num}$ Resultado1)

Para obter ambos os conjuntos de empregados supervisionados nos níveis 1 e 2 por "Fábio Lemos", aplica-se a união: Resultado ← Resultado1 ∪ Resultado2

-Operações de Junção Externa

As operações de Junção Externa são extensões da Junção, são utilizadas quando se deseja manter todas as tuplas de R, ou de S, ou de ambas as relações, no resultado da Junção, caso elas possuam ou não tuplas que se combinem nas relações.
As operações classificam-se em: Junção Externa à Esquerda, denotada por R **=JUN** S, mantém todas as tuplas da relação R; se não há tupla de S que combine, os atributos de S são preenchidos com valores nulos, Junção Externa à Direita, denotada por R **JUN=** S, mantém todas as tuplas da relação S; se não há tupla de R que combine, os atributos de R são preenchidos com valores nulos e Junção Externa Completa: denotada por R **=JUN=** S, mantém todas as tuplas em ambas as relações; se alguma tupla não combina, preenche os atributos referentes com nulos.

Ex.: listar o nome dos empregados e, se for o caso, o nome dos departamentos que eles gerenciam.

Temp ← Empregado **=JUN** $_{NumEmpregado\ =\ NumGerente}$ Departamento

Resultado ← $\pi_{PrimeiroNome,UltimoNome,NomeDepto}$ (Temp)

ou

Temp ← Departamento **JUN=** $_{NumGerente\ =\ NumEmpregado}$ Empregado

Resultado ← $\pi_{PrimeiroNome,UltimoNome,NomeDepto}$ (Temp)

A operação União Externa serve para realizar a união entre tuplas de duas relações, caso as mesmas não sejam compatíveis para união. Tal operação irá encontrar a união entre tuplas de duas relações que são parcialmente compatíveis, significando que apenas alguns atributos são compatíveis para união, os atributos que não são compatíveis para união, de qualquer relação, são mantidos na relação resultante e, caso não possuem valores para uma determinada tupla, seus valores são preenchidos com nulos.

Por exemplo, uma União Externa pode ser aplicada entre as relações AlunoGrad (Nome, Dep, Periodo) e AlunoPos (Nome, Dep, Nivel). A relação resultante é R (Nome, Dep, Periodo, Nivel), uma tupla em R proveniente de tuplas em ambas

relações terá valores para todos os atributos, uma tupla em R proveniente apenas de uma tupla da relação AlunoGrad (aluno de graduação) terá valor nulo para o campo "Nivel", uma tupla em R proveniente apenas de uma tupla da relação AlunoPos (aluno de pós-graduação) terá valor nulo para o campo "Periodo".

-SQL (Structured Query Language)

SQL é uma linguagem de consulta que implementa as operações da álgebra relacional de forma bem amigável, além de permitir a realização de consultas, SQL possibilita: definição da estrutura de dados, definição de restrições de integridade, modificação de dados no banco de dados, especificação de restrições de segurança e controle de transações e utilização em linguagens hospedeiras.

De uma forma geral, SQL utiliza os termos tabela, linha e coluna para relação, tupla e atributo, respectivamente, SQL foi projetada e implementada pela IBM, como uma interface para o sistema de banco de dados relacional SYSTEM R, tendo sido chamada inicialmente de SEQUEL (Structured English QUEry Language). Em 1986, um trabalho conjunto entre o ANSI (American National Standards Institute) e o ISO (International Standards Organization) conduziu a primeira versão padrão de SQL (ANSI 1986), chamada SQL1, em 1992, tal padrão foi revisado e mais expandido, gerando a SQL2, A SQL3 deverá estender a SQL2 com banco de dados orientados a objetos.

-Esquema e Catálogo

A SQL1 não contemplava o conceito de esquema de banco de dados relacional, todas as tabelas eram consideradas parte do mesmo esquema.
Esquema, incorporado à SQL2, é utilizado para agrupar tabelas e outros componentes que pertencem à mesma aplicação de banco de dados, um esquema é definido por um nome e inclui um identificador de autorização para indicar o usuário que é dono do esquema. CREATE SCHEMA Empresa AUTHORIZATION JSilva; os elementos de um esquema incluem tabelas, restrições, visões, domínios e outros componentes.

Catálogo é uma coleção de esquemas num ambiente SQL, esquemas de um mesmo catálogo podem compartilhar certos elementos como, por exemplo, definições de domínio.

-Comando CREATE TABLE

CREATE TABLE é o comando usado para especificar uma nova relação, fornecendo um nome e informando os seus atributos e as suas restrições, inicialmente, os atributos são especificados, informando o nome, o tipo de dado e qualquer restrição para o atributo, como, por exemplo, NOT NULL, depois, são especificadas as restrições (CONSTRAINT) de chave (PRIMARY KEY, UNIQUE) e de integridade referencial (FOREIGN KEY), tais restrições podem ser especificadas também no comando ALTER TABLE, que permite a alteração da definição de uma relação.

CREATE TABLE Empregado

(PrimeiroNome VARCHAR(15) NOT NULL,

InicialMeio CHAR,

UltimoNome VARCHAR(15) NOT NULL,

NumEmpregado CHAR(9) NOT NULL,

DataNascimento DATE,

Endereco VARCHAR(30),

Sexo CHAR,

Salario DECIMAL(10,2),

NumSupervisor CHAR(9),

NumDepto INT NOT NULL,

CONSTRAINT PK_Emp **PRIMARY KEY** (NumEmpregado),

CONSTRAINT FK_NumSup **FOREIGN KEY** (NumSupervisor)

REFERENCES Empregado (NumEmpregado),

CONSTRAINT FK_EmpDep **FOREIGN KEY** (NumDepto)

REFERENCES Departamento (NumDepto)

);

Tipos de dados numéricos: inteiro: integer ou int, smallint, real: float, real, double precision, números formatados: decimal(i, j) ou numeric(i, j) ou dec(i,j) onde "i" é número de dígitos a esquerda e "j" é o número de dígitos a direita do ponto decimal.

Tipos de dados alfanuméricos: cadeia de caracteres de tamanho fixo: char(n) ou character(n), cadeia de caracteres de tamanho variável: varchar(n) ou char varying(n) ou character varying(n), onde "n" é o tamanho máximo de caracteres.

Tipo data: date (aaaa-mm-dd), Tipo hora: time (hh:mm:ss), Pode-se declarar um domínio e usar o nome do domínio como tipo de dado de algum atributo.

CREATE DOMAIN TipoNumEmp **AS** CHAR(9);

Pelo fato da SQL permitir valores nulos como valores de atributos, uma restrição NOT NULL pode ser especificada se o valor nulo não puder ser definido para um atributo.

Para definir um valor default para um atributo, utiliza-se a cláusula DEFAULT <valor> na definição do atributo, o valor default é incluído em uma nova tupla, se um valor explícito não for fornecido para o atributo em questão.

CREATE TABLE Empregado

(. . .

NumDepto INT NOT NULL DEFAULT 1,

. . .

);

Em SQL, pode-se especificar a ação a ser tomada se uma restrição de integridade referencial for violada, mediante à exclusão de uma tupla ou à atualização do valor de uma chave primária. As opções incluem bloqueio (default), propagação (CASCADE), substituição por nulo (SET NULL) e substituição pelo valor default (SET DEFAULT) que devem ser especificadas com a cláusula ON DELETE (em uma operação de exclusão) ou ON UPDATE (em uma operação de atualização), em cada restrição de integridade referencial.

CREATE TABLE Empregado

(PrimeiroNome VARCHAR(15) NOT NULL,

. . .

NumDepto INT NOT NULL DEFAULT 1,

CONSTRAINT PK_Emp **PRIMARY KEY** (NumEmpregado),

CONSTRAINT FK_NumSup **FOREIGN KEY** (NumSupervisor)

REFERENCES Empregado (NumEmpregado)

ON DELETE SET NULL **ON UPDATE** CASCADE,

CONSTRAINT FK_EmpDep **FOREIGN KEY** (NumDepto)

REFERENCES Departamento (NumDepto)

ON DELETE SET DEFAULT **ON UPDATE** CASCADE

);

-Comando DROP SCHEMA

Para remover um esquema de um banco de dados, SQL usa o comando DROP SCHEMA, cuja sintaxe é:

DROP SCHEMA <E> <opção>;

onde <E> é o nome do esquema a ser removido e <opção> pode ser RESTRICT (não elimina o esquema se houver algum elemento nele) ou CASCADE (elimina o esquema e todos os seus elementos). Ex.: **DROP SCHEMA** Empresa **CASCADE**;

-Comando DROP TABLE

Para remover uma relação de um banco de dados, SQL usa o comando DROP TABLE, cuja sintaxe é:

DROP TABLE <R> <opção>;

onde <R> é o nome da relação a ser removida e <opção> pode ser RESTRICT (não elimina a relação se houver alguma restrição ou visão associada a ela) ou CASCADE (elimina a relação e todas as restrições e visões associadas a ela). Ex.: **DROP TABLE** Dependente **CASCADE**;

-Comando ALTER TABLE

O comando ALTER TABLE é usado para adicionar ou remover atributos e restrições de uma relação, e para alterar a definição de um atributo. Possui as seguintes sintaxes:

ALTER TABLE <R> **ADD** <A> <D> : adiciona o atributo <A>, cujo domínio é <D>, na relação existente <R>.

ALTER TABLE Empregado **ADD** Serviço VARCHAR(12);

Neste caso, se já existirem tuplas na relação <R>, o novo atributo receberá valores nulos para essas tuplas.

ALTER TABLE <R> **DROP** <A> <opção> : remove o atributo <A> da relação existente <R>.

A <opção> pode ser RESTRICT (não elimina o atributo se houver alguma restrição ou visão referenciando-o) ou CASCADE (elimina o atributo, as visões e as restrições que o referenciam).

ALTER TABLE Empregado **DROP** Endereco **CASCADE**;

ALTER TABLE <R> **ALTER** <A> **DROP DEFAULT**: remove a cláusula de default referente ao atributo <A> da relação existente <R>.

ALTER TABLE Empregado **ALTER** NumDepto **DROP DEFAULT**;

ALTER TABLE <R> **ALTER** <A> **SET DEFAULT** <V>: adiciona uma cláusula de default, referente ao atributo <A> da relação existente <R>, com o valor <V>.

ALTER TABLE Empregado **ALTER** NumDepto **SET DEFAULT** 2;

ALTER TABLE <R> **DROP CONSTRAINT** <C>: remove a restrição <C> referente à relação existente <R>.

ALTER TABLE Empregado **DROP CONSTRAINT** FK_EmpDep;

ALTER TABLE <R> **ADD CONSTRAINT** <C>: adiciona a restrição <C> na relação existente <R>.

ALTER TABLE Empregado **ADD CONSTRAINT** FK_NumSup **FOREIGN KEY** (NumSupervisor) **REFERENCES** Empregado (NumEmpregado) **ON DELETE** SET NULL **ON UPDATE** CASCADE;

-Consultas Básicas em SQL

A estrutura básica de uma consulta SQL é:

SELECT <A_1>, <A_2>, ... , <A_n> {projeção}

FROM <R_1>, <R_2>, ... , <R_m> {produto cartesiano}

WHERE <cond>; {seleção}

cada <Ai>, para 1 ≤ i ≤ n, representa um atributo, cada <Rj>, para 1 ≤ j ≤ m, representa uma relação, <cond> é uma condição de seleção (expressão lógica), a cláusula WHERE pode ser omitida, se não houver condição de seleção, a lista de atributos A_1,, A_2, ... , A_n pode ser substituída por um asterisco (*) indicando que todos os atributos de todas as relações da cláusula FROM serão projetados. Esta consulta é equivalente à seguinte expressão em álgebra relacional: πA_1, A_2, ...,A_n (σ_{cond} ($R_1 \times R_2 \times R_m$))

Exemplos de consultas básicas:

Número e nome de todos os empregados:

SELECT NumEmpregado, PrimeiroNome, UltimoNome

FROM Empregado;

Nome dos empregados cujo salário seja superior a 500 reais:

SELECT PrimeiroNome, InicialMeio, UltimoNome

FROM Empregado

WHERE Salario > 500;

Empregados de sexo feminino:

SELECT *

FROM Empregado

WHERE Sexo = 'F';

Palavras-chave DISTINCT e ALL

Diferentemente do modelo relacional formal, SQL permite duas tuplas idênticas em uma mesma relação, uma relação em SQL não é um conjunto de tuplas. Para remover as tuplas duplicadas no resultado de uma consulta, deve-se usar a palavra-chave DISTINCT na cláusula SELECT.

SELECT DISTINCT UltimoNome

FROM Empregado;

Para especificar que as tuplas duplicadas não devem ser removidas, deve-se usar a palavra-chave ALL (default).

SELECT ALL UltimoNome

FROM Empregado;

-Operador LIKE

Na SQL, o operador de comparação LIKE permite condições de comparação em partes de uma cadeia de caracteres, o caractere '%' substitui um número arbitrário de caracteres e o caractere '_' substitui um único caractere.

Listar todos os empregados cujo nome começa com 'A':

SELECT *

FROM Empregado

WHERE PrimeiroNome LIKE 'A%';

Listar todos os empregados que nasceram na década de 50:

SELECT *

FROM Empregado

WHERE DataNascimento LIKE '_ _ 5%';

-Operador BETWEEN

Por meio do operador BETWEEN, é possível especificar um intervalo de valores para um atributo. Listar todos os empregados do departamento 5 cujo salário esteja entre 5000 e 8000:

SELECT *

FROM Empregado

WHERE (Salário **BETWEEN** 5000 AND 8000) **AND**

NumDepto = 5;

Valor NULL

SQL permite consultas que verificam se o valor de um atributo é NULL; para isto, utiliza-se o operador IS ou IS NOT.

SELECT PrimeiroNome, UltimoNome

FROM Empregado

WHERE NumSupervisor **IS NULL**;

-Renomeando Relações

Em consultas SQL, é possível "renomear" nomes de relações, usando o operador AS. Obter o nome e o endereço dos empregados que trabalham no departamento 'Pesquisa':

SELECT E.PrimeiroNome, E.UltimoNome, E.Endereco

FROM Empregado **AS** E, Departamento **AS** D

WHERE (E.NumDepto = D.NumDepto) **AND** (D.NomeDepto = 'Pesquisa');

para cada empregado, obter o último nome dos empregados e dos seus supervisores diretos:

SELECT E.UltimoNome, M.UltimoNome

FROM Empregado **AS** E, Empregado **AS** M

WHERE (E.NumSupervisor = M.NumEmpregado);

É possível também "renomear" nomes de atributos.
FROM Empregado **AS** E(PN,IM,UN,Num,DN,End,Sexo,Sal,NumS,NumD)

-Operadores aritméticos

Outra característica da SQL é a permissão em se utilizar operadores aritméticos ('+', '-', '*', '/') nas consultas. Apresentar os salários resultantes de um aumento de 10% a todos os empregados do projeto 'Produto Y':

SELECT PrimeiroNome, UltimoNome, 1.1 * Salario

FROM Empregado, Trabalha_em, Projeto

WHERE (Empregado.NumEmpregado = Trabalha_em.NumEmpregado)

AND (Projeto.NumProj = Trabalha_em.NumProj) **AND** (NomeProj = 'Produto Y');

-Cláusula ORDER BY

A SQL permite ordenar as tuplas no resultado de uma consulta pelos valores de um ou mais atributos, utilizando a cláusula ORDER BY.
SELECT NomeDepto, PrimeiroNome, UltimoNome, NomeProj

FROM Departamento, Empregado, Trabalha_em, Projeto

WHERE (Empregado.NumDepto=Departamento.NumDepto) **AND**

(Empregado.NumEmpregado=Trabalha_em.NumEmpregado)

AND (Projeto.NumProj = Trabalha_em.NumProj)

ORDER BY NomeDepto, PrimeiroNome;

-Operações de Conjunto

A SQL incorporou as seguintes operações de conjunto da álgebra relacional: união (UNION), interseção (INTERSECT) e diferença (EXCEPT). Na operação de união, as tuplas duplicadas são eliminadas automaticamente, para manter as tuplas duplicadas, deve-se usar a cláusula UNION ALL.

SELECT DISTINCT NumProj

FROM Projeto, Departamento, Empregado

WHERE (Projeto.NumDepto=Departamento.NumDepto) **AND** (NumGerente=NumEmpregado) **AND** (UltimoNome= 'Silva')

UNION

SELECT DISTINCT NumProj

FROM Trabalha_em, Empregado

WHERE (Empregado.NumEmpregado=Trabalha_em.NumEmpregado) **AND** (UltimoNome= 'Silva');

Junção entre Tabelas

SQL permite especificar a condição da junção na cláusula FROM, usando a cláusulas JOIN (INNER JOIN) e ON. Obter o nome e o endereço dos empregados que trabalham no departamento 'Pesquisa':

SELECT PrimeiroNome, UltimoNome, Endereco

FROM (Empregado **JOIN** Departamento **ON** Empregado.NumDepto=Departamento.NumDepto)

WHERE NomeDepto= 'Pesquisa';

Pode-se também utilizar a junção natural da álgebra:

SELECT PrimeiroNome, UltimoNome, Endereco

FROM Empregado **NATURAL JOIN** Departamento

WHERE NomeDepto= 'Pesquisa';

Em SQL, é possível utilizar também as operações de junção externa (OUTER JOIN) da álgebra relacional.

SELECT E.UltimoNome AS NomeEmp, M.UltimoNome AS NomeSup

FROM (Empregado **AS** E **LEFT OUTER JOIN** Empregado **AS** M

ON E.NumSupervisor = M.NumEmpregado);

O exemplo exemplificou uma junção externa à esquerda (LEFT OUTER JOIN); mas existem também a junção externa à direita (RIGHT OUTER JOIN) e a junção externa completa (FULL OUTER JOIN).

-Subconsultas

Uma subconsulta é um bloco completo (SELECT ... FROM ... WHERE) que existe dentro da cláusula WHERE de uma outra consulta, chamada consulta externa.

SELECT DISTINCT NumProj

FROM Projeto

WHERE NumProj **IN** (**SELECT** NumProj

FROM Projeto, Departamento, Empregado

WHERE (NumGerente=NumEmpregado) **AND**

(Projeto.NumDepto=Departamento.NumDepto)

AND (UltimoNome= 'Silva'))

OR

NumProj **IN** (SELECT T.NumProj

FROM Trabalha_em **AS** T, Empregado **AS** E

WHERE (E.NumEmpregado=T.NumEmpregado)

AND (E.UltimoNome= 'Silva'));

-Operadores SOME e ALL

Além do operador IN, existem outros operadores que são usados para comparar um valor 'v' com um conjunto 'V': <operador lógico> SOME (ou ANY): retorna verdadeiro se 'v' é <operador lógico> que algum valor do conjunto 'V'; <operador lógico> ALL: retorna verdadeiro se 'v' é <operador lógico> que todos os valores do conjunto 'V', os operadores SOME e ALL precisam acompanhar algum operador lógico: =, <>, >, >=, <, <=, o operador '= SOME' é equivalente ao operador IN.

SELECT PrimeiroNome, UltimoNome

FROM Empregado

WHERE Salario > **ALL** (**SELECT** Salario **FROM** Empregado

WHERE NumDepto=5);

SELECT E.PrimeiroNome, E.UltimoNome

FROM Empregado **AS** E

WHERE E.Salario > **SOME** (**SELECT** M.Salario

FROM Empregado **AS** M, Departamento **AS** D **WHERE** (M.NumDepto=D.NumDepto) **AND** (D.NomeDepto= 'Pesquisa'));

-Conjuntos Explícitos de Valores

Em SQL, é possível utilizar um conjunto explícito de valores na cláusula WHERE ao invés de uma subconsulta.

SELECT DISTINCT NumEmpregado

FROM Trabalha_em

WHERE NumProj IN (1, 2, 3);

-Subconsultas Correlacionadas

Sempre que uma condição na cláusula WHERE de uma subconsulta faz referência a algum atributo de uma relação declarada na consulta externa, diz-se que as duas consultas são correlacionadas, neste caso, a subconsulta é avaliada uma vez para cada tupla (ou combinação de tuplas) da consulta externa.

SELECT E.PrimeiroNome, E.UltimoNome

FROM Empregado **AS** E

WHERE E.NumEmpregado **IN** (**SELECT** D.NumEmpregado

FROM Dependente AS D

WHERE E.PrimeiroNome = NomeDependente);

-Função EXISTS

A função EXISTS é utilizada para verificar se o resultado de uma subconsulta correlacionada é vazio ou não.

SELECT E.PrimeiroNome, E.UltimoNome

FROM Empregado **AS** E

WHERE NOT EXISTS (**SELECT** * **FROM** Dependente **AS** D

WHERE E.NumEmpregado = D.NumEmpregado);

SELECT E.PrimeiroNome, E.UltimoNome

FROM Empregado AS E

WHERE EXISTS (**SELECT** * **FROM** Dependente **AS** D

WHERE E.NumEmpregado=D.NumEmpregado)

AND

EXISTS (**SELECT** * **FROM** Departamento

WHERE E.NumEmpregado = NumGerente);

-Agrupamento e Funções de Agregação

A linguagem SQL incorpora os conceitos de agrupamento e funções de agregação definidos na álgebra relacional. As principais funções de agregação são: COUNT, número de tuplas recuperadas em uma consulta, SUM, soma dos valores de um atributo em uma consulta, MAX, valor máximo de um atributo em uma consulta, MIN, valor mínimo de um atributo em uma consulta, AVG, média dos valores de um atributo em uma consulta.

As funções de agregação podem ser usadas em uma cláusula SELECT ou em uma cláusula HAVING, a cláusula GROUP BY serve para agrupar tuplas que possuem o mesmo valor para os atributos relacionados em tal cláusula. A cláusula HAVING pode ser usada em conjunto com a cláusula GROUP BY para especificar uma condição de seleção a ser aplicada em cada grupo de tuplas recuperadas em uma consulta, somente os grupos que satisfizerem a condição serão retornados.

SELECT SUM(Salario), **MAX**(Salario), **MIN**(Salario), **AVG**(Salario)

FROM (Empregado **JOIN** Departamento **ON** Empregado.NumDepto=Departamento.NumDepto)

WHERE NomeDepto= 'Pesquisa';

SELECT COUNT(*)

FROM Empregado **AS** E, Departamento **AS** D

WHERE (E.NumDepto = D.NumDepto) **AND** (D.NomeDepto = 'Pesquisa');

SELECT COUNT (DISTINCT Salario)

FROM Empregado;

SELECT PrimeiroNome, UltimoNome

FROM Empregado **AS** E

WHERE (**SELECT COUNT**(*)

FROM Dependente AS D

WHERE E.NumEmpregado=D.NumEmpregado) >= 2;

SELECT NumDepto, **COUNT**(*), **AVG**(Salario)

FROM Empregado **GROUP BY** NumDepto;

```
SELECT P.NumProj, P.NomeProj, COUNT(*)
FROM Projeto AS P, Trabalha_em AS T
WHERE P.NumProj=T.NumProj
GROUP BY P.NumProj, P.NomeProj;

SELECT P.NumProj, P.NomeProj, COUNT(*)
FROM Projeto AS P, Trabalha_em AS T
WHERE P.NumProj=T.NumProj
GROUP BY P.NumProj, P.NomeProj
HAVING COUNT(*) > 2;


SELECT D.NumDepto, COUNT(*)
FROM Empregado AS E, Departamento AS D
WHERE E.NumDepto=D.NumDepto AND E.Salario>5000 AND
E.NumDepto IN ( SELECT NumDepto
                FROM Empregado
                GROUP BY NumDepto
                HAVING COUNT(*) > 5)
GROUP BY D.NumDepto;
```

-Comando INSERT

Em sua forma mais simples, o comando INSERT é utilizado para adicionar uma única tupla a uma relação, deve-se especificar o nome da relação e uma lista de valores para a tupla, os valores da tupla devem estar relacionados na mesma ordem em que foram definidos no CREATE TABLE.

```
INSERT INTO Empregado
VALUES ('Rosana', 'P', 'Souza', '653298653', '1962-12-30', 'R.Alagoas, 1230',
'F', 3000, '987654321', 4);
```

Uma segunda forma permite especificar nomes explícitos de atributos que correspondem aos valores fornecidos no comando INSERT.

```
INSERT INTO Empregado(PrimeiroNome, UltimoNome, NumEmpregado,
NumDepto)
VALUES ('Rosana', 'Souza', '653298653', 4);
```

Uma variação do comando INSERT inclui múltiplas tuplas numa relação, conjuntamente com a criação da relação e a carga da mesma com o resultado de uma consulta.

CREATE TABLE Info_Deptos

(Nome_Depto **VARCHAR**(15),

Num_de_Emps **INTEGER**,

Total_Sal **INTEGER**);

INSERT INTO Info_Deptos

SELECT NomeDepto, **COUNT**(*), **SUM**(Salario)

FROM (Departamento **JOIN** Empregado **ON** Departamento.NumDepto=Empregado.NumDepto)

GROUP BY NomeDepto;

-Comando DELETE

O comando DELETE remove tuplas de uma relação, possui a cláusula WHERE para selecionar as tuplas a serem excluídas, quando não especificada, todas as tuplas da relação determinada são excluídas; entretanto, a tabela permanece no banco de dados como uma tabela vazia, as tuplas são explicitamente excluídas de uma só relação. Entretanto, a exclusão pode se propagar para tuplas de outras relações de acordo com as restrições de integridade referencial definidas.

DELETE FROM Empregado

WHERE UltimoNome = 'Silva';

DELETE FROM Empregado

WHERE Salario < 800;

DELETE FROM Empregado

WHERE NumDepto **IN** (**SELECT** NumDepto

FROM Departamento

WHERE NomeDepto = 'Pesquisa');

DELETE FROM Empregado;

-Comando UPDATE

O comando UPDATE é utilizado para modificar valores de atributos de uma ou mais tuplas em uma única relação, cláusula WHERE para selecionar as tuplas a serem modificadas, cláusula SET para especificar os atributos a serem

modificados e o seus novos valores, a modificação do valor de uma chave primária pode se propagar para chaves estrangeiras de outras relações, de acordo com as restrições de integridade referencial definidas.

Alterar a localização e o número do departamento controlador do projeto número 10 para 'Anchieta' e 1, respectivamente:

UPDATE Projeto

SET Localização = 'Anchieta', NumDepto = 1

WHERE NumProj = 10;

Diversas tuplas podem ser modificadas com um único comando UPDATE.
Aumentar em 10% os salários de todos os empregados do Departamento 'Pesquisa':

UPDATE Empregado

SET Salario = Salario * 1.1

WHERE NumDepto **IN** (**SELECT** NumDepto

FROM Departamento

WHERE NomeDepto='Pesquisa');

-Visões em SQL

Uma visão em SQL é uma relação única que é derivada de outras relações ou de outras visões previamente definidas, uma visão não existe necessariamente na forma física: é considerada uma relação virtual, limitando assim, as operações de atualização que podem ser aplicadas, não há limitação na consulta de uma visão.
O comando para especificar uma visão é CREATE VIEW, deve-se especificar o nome da visão, uma lista de nomes de atributos e uma consulta para especificar o conteúdo da visão, se nenhum dos atributos da visão resultar da aplicação de funções ou operações aritméticas, não há necessidade de especificar os nomes de atributos para a visão, já que seriam os mesmos nomes de atributos das relações definidoras da visão.

CREATE VIEW V_Trabalha_em

AS SELECT PrimeiroNome, UltimoNome, NomeProj, Horas

FROM Empregado **AS** E, Projeto **AS** P, Trabalha_em **AS** T

WHERE (E.NumEmpregado = T.NumEmpregado) **AND** (P.NumProj = T.NumProj);

CREATE VIEW V_Info_Deptos (NomeDepto, QteEmps, TotalSal)

AS SELECT NomeDepto, COUNT(*), SUM(Salario)

FROM (Departamento **JOIN** Empregado **ON** Departamento.NumDepto=Empregado.NumDepto)

GROUP BY NomeDepto;

Uma vez definida uma visão, pode-se realizar consultas utilizando a mesma. Obter o primeiro e o último nome de todos os empregados que trabalham no projeto 'Projeto X':

SELECT PrimeiroNome, UltimoNome

FROM V_Trabalha_em

WHERE NomeProj = 'Projeto X';

Uma das principais vantagens de uma visão é simplificar a especificação de certas consultas. As visões também são utilizadas como mecanismos de segurança e autorização. Uma visão está sempre atualizada, pois não efetua alguma ação no momento em que é definida, e sim no momento em que se especifica uma consulta utilizando-a. Para remover uma visão, utiliza-se o comando DROP VIEW.

DROP VIEW V_Trabalha_em;

Com relação à atualização de dados das relações definidoras de uma visão, a partir da mesma, pode-se dizer que: uma visão definida a partir de uma única relação é atualizável se os atributos da visão contiverem a chave primária (ou possivelmente alguma outra chave candidata) de tal relação, pois isso mapeia cada tupla da visão para uma única tupla da relação definidora, uma visão definida a partir de múltiplas relações utilizando junções geralmente não é atualizável, pois a atualização pela visão deve ser mapeada em diversas atualizações nas relações definidoras para fornecer o efeito desejado, uma visão definida com o uso de agrupamento e funções agregadas não é atualizável, pois alguns atributos não existem nas relações definidoras.

-Conexão entre PostgreSQL e C++

Exemplo de código:

```
#include <iostream>

#include "libpq-fe.h" //biblioteca necessária

using namespace std;

int main()

{

        int nTuplas, nAtributos;

        //declara uma variavel logica que representara o

        //o banco de dados
```

```
PGconn *conn = NULL;
//Cria uma conexao com um banco do PostgreSQL
conn = Pqconnectdb ("host=localhost port=5432 dbname=empresa user=postgres password=postgres");
//Verifica a conexao feita
if (PQstatus(conn) != CONNECTION_OK){
        //Retorna uma mensagem de erro do PostgreSQL
        cout << PQerrorMessage(conn);
        //Encerra a conexao com o banco de dados,
        //liberando a memoria
        PQfinish(conn);
        return 1;
}
else
cout << "Conexao executada com sucesso!" << endl;
//Declara um ponteiro para resultado de uma consulta
PGresult *res;
//Faz uma consulta ao banco de dados
res = PQexec(conn, "select * from funcionario;");
//Verifica a validade da consulta
if (PQresultStatus(res) != PGRES_TUPLES_OK){
        cout << PQerrorMessage(conn);
        PQclear(res);
        PQfinish(conn);
        return 2;
}
// Obtem o numero de tuplas
nTuplas = PQntuples(res);
// Obtem o numero de atributos
nAtributos = PQnfields(res);
// Percorre todos as tuplas
```

```
        for (int i = 0; i < nTuplas; i++){
                // Percorre todos os atributos
                for (int j = 0; j < nAtributos; j++){
                        // Imprime o valor do atributo j da tupla i
                        cout << PQgetvalue(res, i, j) << "\t";
                }
                cout << endl;
        }
        // Fecha o acesso ao resultado da consulta
        PQclear(res);
        // Fecha a conexao com o banco de dados
        PQfinish(conn);
```

Compilação:

g++ -o exemplo -I /usr/include/postgresql/ exemplo.cpp -L /usr/lib/ -lpq

Considerando que:

• o SGBD eh o PostgreSQL 9.2;

• o arquivo libpq-fe.h encontra-se no diretório "/usr/include/postgresql";

• a biblioteca libpq encontra-se no diretório "/usr/lib".