**SQL DDL** (*Linguagem de Definição de Dados*) – Comandos para a definição, a modificação e a remoção de relações, além da criação e da remoção de índices.

*Linguagem de Manipulação de Dados Embutida*: pode ser utilizada a partir de linguagens de programação de propósito geral

*Definição de visões*: inclui comandos para a criação e a remoção de visões

*Restrição de integridade*: possui comandos para a especificação de restrições de integridade

*Autorização*: inclui comandos para a especificação de direitos de acesso a relações e visões

*Gerenciamento de transações*: introduz comandos para especificação do início e do fim das transações.

*Recuperação de falhas*: introduz comandos para utilização do arquivo de *log.*

**CREATE DATABASE | SCHEMA** – cria um esquema de BD relacional

CREATE {DATABASE |SCHEMA} nome

[USER `username` [PASSWORD `password`]]

... ;

Agrupa as tabelas e outros comandos que pertencem à mesma aplicação;

Identifica o proprietário do esquema

Característica – o esquema inicial não possui tabelas/dados

**DROP DATABASE | SCHEMA** – remove um esquema de BD relacional

DROP DATABASE {DATABASE | SCHEMA} nome

[CASCADE | RESTRICT];

Remove um esquema de BD relacional

- tabelas/dados

- índices

- arquivos de log

Usuários autorizados

- proprietário do banco de dados

- DBA ou usuário com privilégio de *root*

CASCADE – remove um esquema de BD, incluindo todas as suas tabelas e os seus outros elementos

RESTRICT – remove um esquema de BD somente se não existirem elementos definidos para esse esquema

**CREATE TABLE** – cria uma nova tabela (relação) no BD, a nova tabela não possui dados

CREATE TABLE nome\_tabela ( A1 D1 R1,

A2 D2 R2,

...

An Dn Rn );

- nome do atributo: Ai (1 <= i <= n)

- tipo de dado (domínio do atributo): Di

- restrições que atuam no atributo: Ri

*Exemplos de tipos de dados*:

Numéricos smallint | integer | float | double precision

decimal | numeric

Hora/Data date | time | timestamp

Strings char | character | varchar | ...

Outros blob

*Restrições de Integridade*:

Valor nulo

* Representado por NULL
* Membro de todos os domínios

Restrição NOT NULL

* Especificada quando NULL não é permitido
* Proíbe que o atributo receba valor nulo

Comparações

* usar IS NULL e IS NOT NULL

Cláusula **PRIMARY KEY**

* Identifica os atributos da relação que foram a sua chave primária
  + Os atributos devem ser definidos como NOT NULL
* Sintaxe
  + PRIMARY KEY (atributo1, atributo2, ..., atributox)

Cláusula **UNIQUE**

* Não permite valores duplicados para um determinado atributo

Cláusula **DEFAULT**

* Associa um valor *default* para um atributo, caso nenhum outro valor seja especificado

Cláusula **CHECK**

* Especifica um predicado que precisa ser satisfeito por todas as tuplas de uma relação
* Exemplos:

saldo int CHECK (saldo>=0)

nível char(15) CHECK (nível IN (`Bacharelado`,`Mestrado`,Doutorado`))

Integridade referencial

* Dependência existente entre a chave estrangeira de uma relação e a chave primária da relação relacionada
* Problemas
  + - * Atualização ou exclusão de elementos da chave primária sem fazer um ajuste coordenado nas chaves estrangeiras
      * Inclusão ou alteração de valores não nulos na chave estrangeira que não existam na chave primária

Cláusula **FOREIGN KEY**

* Características
  + Elimina a possibilidade de violação da integridade referencial
  + Reflete nas chaves estrangeiras todas as alterações na chave primária
* Sintaxe:

FOREIGN KEY (atributos)

REFERENCES nome\_relação (atributos)

[ON UPDATE [NO ACTION | CASCADE | SET NULL | SET DEFAULT]]

[ON DELETE [NO ACTION | CASCADE | SET NULL | SET DEFAULT]]

**DROP TABLE** - remove uma tabela (relação) e todas as suas instâncias do BD: dados, índices, gatilhos que referenciam a tabela, metadados

**ALTER TABLE** -altera a estrutura de uma tabela (relação) já existente no BD:

Adiciona, remove, altera (colunas ou restrições de integridade)

Inclui novas colunas na tabela: ALTER TABLE nome\_tabela

ADD (A1 D1 R1),

...

ADD(An Dn Rn)

Elimina uma coluna já existente da tabela: ALTER TABLE nome\_tabela DROP A1

Modifica o nome de uma coluna existente de A1 para A2: ALTER TABLE nome\_tabela ALTER [COLUMN] A1 TO A2

Modifica o tipo de dado de uma coluna: ALTER TABLE nome\_tabela

ALTER [COLUMN] A1 TYPE SMALLINT

**CREATE DOMAIN** – cria um domínio para um tipo de dados

**DROP DOMAIN** – remove um domínio existente do BD

**ALTER DOMAIN** – altera a definição de domínio

**SQL DML** (*Linguagem de Manipulação de Dados*) – Comandos para a consulta, a inserção, a remoção e a modificação de tuplas no banco de dados.

**SELECT ... FROM ... WHERE ...** – lista atributos de uma ou mais tabelas de acordo com alguma condição

SELECT <lista de atributos e funções>

FROM <lista de tabelas>

[WHERE predicado]

[GROUP BY <atributos de agrupamento>]

[HAVING <condição para agrupamento>]

[ORDER BY <lista de atributos>];

SELECT – lista os atributos e/ou as funções a serem exibidos no resultado da consulta: operação de projeção.

FROM – especifica as relações a serem examinadas na avaliação da consulta: operação de produto cartesiano.

WHERE – Especifica as condições para a seleção das tuplas no resultado da consulta:

As condições devem ser definidas sobre os atributos das relações que aparecem na cláusula FROM

Inclui condição de junção

Corresponde ao predicado de seleção da álgebra relacional

Pode ser omitida

WHERE <atributo> <operador>

<valor | atributo | lista de valores>

* Operador:
  + Conjunção de condições: AND
  + Disjunção de condições: OR
  + Negação de condições: NOT
* Operadores de comparação

|  |  |  |  |
| --- | --- | --- | --- |
| Igual a | = | Diferente de | < > |
| Maior que | > | Maior ou igual a | >= |
| Menor que | < | Menor ou igual a | <= |
| Entre dois valores | BETWEEN ...  AND | De cadeias de caracteres | LIKE ou  NOT LIKE |

* Precedência: NOT; operadores de comparação; AND; OR
* Operadores de comparação de cadeias de caracteres
  + % (porcentagem): substitui qualquer *string*
  + \_ (underscore): substitui qualquer *caractere*
* Característica: operadores sensíveis ao caso (letras maiúsculas são diferentes de minúsculas)

Exemplos:

Qualquer *string* que se inicie com ‘Mar’: WHERE nome\_região LIKE ‘Mar%’

Qualquer *string* de 4 caracteres que se inicie com ‘Mar’: WHERE nome\_região LIKE ‘Mar\_’

Operação de Conjuntos – as relações participantes das operações **precisam ser *compatíveis***.

Operações oferecidas dependem do SGBD

UNION – une todas as linhas selecionadas por duas consultas, **eliminando** as linhas duplicadas

UNION ALL – une todas as linhas selecionadas por duas consultas, **inclusive** as linhas duplicadas

Oracle 9i SQL

UNION – une todas as linha selecionadas por duas consultas, eliminando as linhas duplicadas

UNION ALL – une todas as linhas selecionadas por duas consultas, inclusive as linhas duplicadas

INTERSECT – retorna as linha selecionadas tanto pela 1ª consulta quanto pela 2ª consulta, eliminando linhas duplicadas que aparecem na resposta final

MINUS – retorna as linhas selecionadas pela 1ª consulta que não foram selecionadas pela 2ª consulta, eliminando linhas duplicadas que aparecem na resposta final

Junção Natural – SQL (primeiras versões): não tem uma representação p/ a operação de junção

Definida em termos de: um produto cartesiano, uma seleção e uma projeção.

Junção – não é representada explicitamente:

Cláusulas SELECT e WHERE – especificam atributos com mesmo nome usando o nome da tabela e o nome do atributo (nome\_tabela.nome\_atributo)

Cláusula FROM – possui mais do que uma tabela

Cláusula WHERE – inclui as condições de junção

Exemplos: SELECT nome\_vinícula, nome\_região

FROM vinícula, região

WHERE vinícola.região\_id = região.região\_id;

SELECT nome\_vinícpla, nome\_região, nome\_vinho

FROM vinícola, região, vinho

WHERE vinícola.região\_id = região.região\_id

AND vinho.vinícola\_id = vinícola.vinícola\_id;

Cláusula AS – renomeia:

*Atributos* – deve aparecer na cláusula SELECT, útil para visualização das respostas na tela

*Relações* – deve aparecer na cláusula FROM, útil quando a mesma relação é utilizada mais do que uma vez na mesma consulta

Exemplo: SELECT nome\_vinícula AS nome\_da\_vinícola,

nome\_região AS localizada\_na\_região,

nome\_vinho AS produz\_o\_vinho

FROM vinícola AS V,

região AS R,

vinho AS Vi

WHERE V.região\_id = R.região\_id

AND Vi.vinícola\_id = V.vinícola\_id;

Cláusula ORDER BY – ordena as tuplas que aparecem no resultado de uma consulta:

- asc (padrão): ordem ascendente - desc: ordem descendente

Ordenação pode ser especificada em vários atributos: a ordenação referente ao primeiro atributo é prioritária. Se houver valores repetidos, então é utilizada a ordenação referente ao segundo atributo, e assim por diante.

Exemplo: Liste os dados da relação vinícola. Ordene o resultado pelo nome da vinícola em ordem descendente e pela região da vinícola em ordem ascendente.

SELECT \*

FROM vinícola, região

WHERE vinícola.região\_id = região.região\_id

ORDER BY nome\_vinícola desc,

nome\_região asc

Função de Agregação

Função: - média 🡺 AVG( )

- mínimo 🡺 MIN( )

- máximo 🡺 MAX( )

- total 🡺 SUM( )

- contagem 🡺 COUNT( )

Observação:

- DISTINCT: não considera valores duplicados

- ALL: inclui valores duplicados

Características: recebem uma coleção de valores como entrada, retornam um único valor

Entrada:

sum( ) e avg( ): conjunto de números

demais funções: tipos de dados numéricos e não-numéricos

Resultado de uma consulta:

* Ordem de apresentação dos atributos
  + Ordem dos atributos na cláusula SELECT
* Ordem de apresentação dos dados
  + Ordem ascendente ou descendente de acordo com a cláusula ORDER BY
  + Sem ordenação
* Duas ou mais tuplas podem possuir valores idênticos de atributos
  + Eliminação de tuplas duplicadas: SELECT DISTINCT

**INSERT INTO ...**– insere dados em uma tabela

Realizada através de especificação:

- De uma tupla particular

- De uma consulta que resulta em um conjunto de tuplas a serem inseridas

Valores dos atributos das tuplas inseridas devem pertencer ao domínio do atributo

Atributos sem valores: especificados por NULL ou valor DEFAULT

Ordem dos atributos deve ser mantida INSERT INTO nome\_tabela

VALUES(V1, V2, ..., Vn);

Ordem dos atributos não precisa ser mantida INSERT INTO nome\_tabela (A1, A2, ..., An)

VALUES (V1, V2, ..., Vn);

Tuplas resultantes da cláusula SELECT serão inseridas na tabela nome\_tabela: INSERT INTO nome\_tabela

SELECT ...

FROM ...

WHERE ... ;

**DELETE FROM ... WHERE** – remove dados de tabelas já existentes

Remove tuplas inteiras

Opera apenas em uma relação

Tuplas de mais de uma relação a serem removidas: um comando DELETE para cada relação

* A remoção de uma tupla de uma relação deve ser propagada para tuplas em outras relações devido às restrições de integridade referencial

DELETE FROM nome\_tabela

WHERE predicado;

Cláusula WHERE: é opcional – todas as tuplas da tabela são eliminadas, a tabela continua a existir

Predicado: pode ser complexo

Exemplos: Remove a tupla referente a vinícola\_id = 10 (tabela vinícula, tabela vinho se a opção CASCADE foi especificada na cláusula ON DELTE do campo vinícola\_id desta tabela): DELETE FROM vinícola

WHERE vinícola\_id = 10;

Remove todos os dados da tabela região: DELETE FROM região

**UPDATE ... SET ... WHERE** – altera dados específicos de uma tabela, opera apenas em uma relação

* Atualização da chave primária deve ser propagada para tuplas em outras relações devido às restrições de integridade referencial

UPDATE nome\_tabela

SET coluna = <valor>

WHERE predicado;

Cláusula WHERE: é opcional

Exemplos de <valor>: NULL, ‘string’, UPPER ‘string’

Exemplos: Alterar os anos de produção de vinhos de 2007 para 2003: UPDATE vinho

SET ano\_vinho = 2003

WHERE ano\_vinho = 2007;

Suponha o atributo adicional *preço* na tabela vinho. Aumentar os preços dos vinhos em 10%

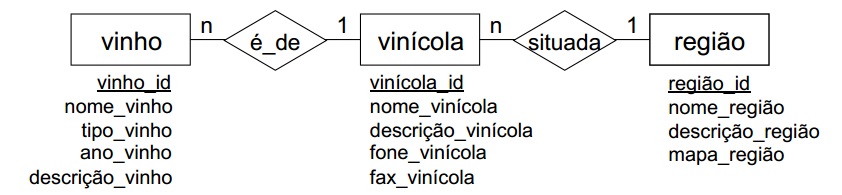
UPDATE vinho SET preço = preço \* 1.10;

Alterar o valor de vinícola\_id = 10 para vinícola\_id = 2 (tabela vinícola, tabela vinho se a opção CASCADE foi especificada na cláusula ON UPDATE do campo vinícola\_id desta tabela): UPDATE vinícola

SET vinícola\_id = 10

WHERE vinícola\_id = 2;

EXEMPLOS



região (região\_id, nome\_região, mapa\_região, descrição\_região)

vinícola(vinícola\_id, nome\_vinícola, descrição\_vinícola, fone\_vinícola, fax\_vinícola, *região\_id*)

vinho(vinho\_id, nome\_vinho, tipo\_vinho, ano\_vinho, descrição\_vinho, *vinícola\_id*)

CREATE DATABASE loja\_vinhos;

CREATE TABLE região (

região\_id smallint DEFAULT ‘0’ NOT NULL,

nome\_região varchar(100) DEFAULT ‘ ‘ NOT NULL,

mapa\_região blob,

descrição\_região blob,

PRIMARY KEY (região\_id),

);

CREATE TABLE vinícola (

vinícola\_id smallint NOT NULL,

nome\_vinícola varchar(100) DEFAULT ‘ ‘ NOT NULL,

descrição\_vinícola blob,

fone\_vinícola varchar(15),

fax\_vinícola varchar(15),

região\_id smallint DEFAULT ‘0’ NOT NULL,

PRIMARY KEY (vinícola\_id),

FOREIGN KEY (região\_id)

REFERENCES região(região\_id)

ON UPDATE SET DEFAULT,

ON DELETE SET DEFAULT,

);

CREATE TABLE vinho (

vinho\_id smallint NOT NULL,

nome\_vinho varchar(50) DEFAULT ‘ ’ NOT NULL,

tipo\_vinho varchar(10) DEFAULT ‘ ‘ NOT NULL,

ano\_vinho integer DEFAULT ‘0’ NOT NULL,

descrição\_vinho blob,

vinícola\_id smallint DEFAULT ‘0’ NOT NULL,

PRIMARY KEY (vinho\_id),

FOREIGN KEY (vinícola\_id)

REFERENCES vinícola(vinícola\_id),

ON UPDATE CASCADE,

ON DELETE CASCADE,

);

SELECT \*

FROM região;

SELECT região\_id, nome\_região

FROM região

WHERE nome\_região LIKE ‘M%’AND

região\_id >= 3 AND

mapa\_região IS NOT NULL;

Liste os anos de fabricação dos vinhos para vinhos tintos e bancos

SELECT ano\_vinho

FROM vinho

WHERE tipo\_vinho = ‘tinto’

UNION

SELECT ano\_vinho

FROM vinho

WHERE tipo\_vinho = ‘branco’;

**Funções de Agregação** – vinho(vinho\_id, nome\_vinho, tipo\_vinho, preço, vinícola\_id)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vinho\_id | Nome\_vinho | Tipo\_vinho | preço | Vinícola\_id |
| 10 | Amanda | Tinto | 100.00 | 1 |
| 09 | Belinha | Branco | 200.00 | 1 |
| 05 | Camila | Rosê | 300.00 | 1 |
| 15 | Daniela | Branco | 250.00 | 2 |
| 27 | Eduarda | Branco | 150.00 | 2 |
| 48 | Fernanda | Tinto | 7.00 | 2 |
| 13 | Gabriela | Tinto | 397.00 | 3 |
| 12 | Helena | branco | 333.00 | 3 |

Qual a média dos preços? SELECT AVG(preço)

R: 217.125 FROM vinho

Qual a soma dos preços? SELECT SUM(preço)

R: 1737.00 FROM vinho

Qual o preço mais baixo? SELECT MIN(preço)

R: 7.00 FROM vinho

Qual o preço mais alto? SELECT MAX(preço)

R: 397.00 FROM vinho

Quantos vinhos existem na relação vinho? SELECT COUNT(vinho\_id)

R: 8 FROM vinho

Quantos tipos de vinho *diferentes* existem na relação vinhos? SELECT COUNT(DISTINCT tipo\_vinho)

R: 3 FROM vinho

**BD Geográfico** - armazenam *dados* / *objetos georreferenciados* da superfície terreste;

Manipulam *grandes volumes* de informações de *grande complexidade*;

Possuem atributos convencionais e *atributos não convencionais*.

Oferece a análise e consultas espaciais.

É possível calcular: área, tamanho e centroide de um objeto; Distâncias entre dois objetos; União e interseção entre objetos; Entre outras operações.

Um BDG geralmente é capaz de responder as questões como: Quais os estados adjacentes ao estado de São Paulo? Quais as rodovias cortam o município de São Paulo? Qual a distância entre BH e Brasília?

Um dado geográfico: descreve uma determinada localização ou forma. Descreve *objetos* ou *fenômenos* que acontecem na superfície terrestre e que possui uma *posição geográfica*. Exemplos: estrada, rio, floresta, hospital, ...

Sistema de Informação Geográfica (SIG): sistemas que realizam o tratamento computacional de dados geográficos.

Funcionalidades como: entrada e validação de dados espaciais; armazenamento e gerenciamento; saída e apresentação visual; transformação de dados espaciais; interação com o usuário; ferramentas para consulta e análise espacial.

Tipos de Dados Vetoriais:

Ponto (zero-dimensional): um **único ponto** na superfície terrestre.

Propriedades (coordenadas): X – Longitude Y – Latitude

Ex.: Em um mapa em uma cidade, um ***Point***pode representar um hospital.

Em SQL: SELECT ‘POINT (3 4)’

Múltiplos Pontos: é uma coleção de pontos na mesma dimensão. Os pontos não estão conectados.

Ex.: Em um mapa de uma cidade, um ***MultiPoint*** pode representar as paradas de ônibus.

Em SQL: SELECT ‘MULTIPOINT(0 0, 1 2)’

Linha (unidimensional): é uma coleção de pontos na mesma dimensão que estão conectados. Uma linha deve conter pelo menos dois pontos diferentes.

Ex.: Num mapa de uma determinada cidade uma ***LineString*** pode presentar uma rua.

Em SQL: SELECT ‘LINESTRING(0 0, 1 1, 1 2)’

Múltiplas Linhas: é uma coleção de linhas.

Ex.: Em um mapa regional, uma ***MultiLineString*** pode representar uma malha de rodovias.

Em SQL: SELECT ‘MULTILINESTRING((0 0, 1 1, 1 2), (2 3, 3 2, 5 4))’

Polígono (bidimensional): composto por uma linha fechada. Um polígono deve conter pelo menos três pontos distintos.

Ex.: Num mapa de um país um ***Polygon*** pode representar um estado.

Em SQL: SELECT ‘POLYGON((0 0, 4 0, 4 4, 0 4, 0 0))’

Ex.: Um polígono pode conter *buracos*

Em SQL: SELECT ‘POLYGON((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1))’

Múltiplos Polígonos: é uma coleção de polígonos.

Ex.: Em um mapa regional, um ***MultiPolygon*** pode representar um sistema de lagos.

Em SQL: SELECT ‘MULTIPOLYGON(((-1 -1, -1 -2, -2 -1, -1 -1)), ((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 2 1, 2 2, 1 2, 1 1)))’

Coleção de Geometrias: formado por *qualquer tipo de dado espacial* (ponto, linha, polígono)

Ex.: Num mapa de um país um ***GeometryCollection*** pode representar um estado e a sua capital.

Em SQL: SELECT ‘GEOMETRYCOLLECTION (POINT(2 3), LINESTRING(2 3, 3 4))’

Criando um BD Geográfico : CREATE DATABASE bd\_geo

TEMPLATE = template\_postgis\_21

Excluindo um BD Geográfico: DROP DATABASE bd\_geo

Criando uma tabela geográfica: CREATE TABLE escola(id serial not null, nome varchar(100), geometria geometry)

Alterando uma tabela geográfica: ALTER TABLE escola ADD COLUMN geom2 geometry

Excluindo uma tabela geográfica: DROP TABLE escola

Criando e Inserindo Dados Geográficos (Pontos, Linhas e Polígonos):

CREATE TABLE cidade(id serial not null,

nome varchar(100),

sede\_pref geometry,

rua geometry,

area geometry);

INSERT INTO cidade(nome, sede\_pref, rua, area)

VALUES (‘São Carlos’, ‘POINT(2 1)’, ‘LINESTRING(1 1, 3 3)’, ‘POLYGON((1 1, 4 1, 1 4, 1 1))’);

Recuperando Dados Geográficos: SELECT nome, ST\_AsText(sede\_pref), ST\_AsText(rua), ST\_AsText(área)

FROM cidade;

Atualizando dados geográficos: UPDATE cidade

SET nome = ‘São Carlos’, sede\_pref = ‘POINT(3 3)’

WHERE id = ‘1’;

Excluindo dados geográficos: DELETE FROM cidade where id = ‘1’

/\* Liste as informações dos empregados que trabalham para o departamento 4 e que recebem salário maior do que R$25.000,00 ou que trabalham para o departamento 5 e que recebem salário maior do que R$30.000,00.\*/

SELECT \*

FROM empregado

WHERE (nro\_departamento = 4 AND salário\_emp > 25000)

OR (nro\_departamento = 5 AND salário\_emp > 30000);

/\* Liste o primeiro nome, o último nome e o salário dos empregados que trabalham para o departamento 4 e que recebem salário maior do que R$25.000,00. \*/

SELECT primeiro\_nome\_emp, último\_nome\_emp, salário\_emp

FROM empregado

WHERE nro\_departamento = 4 AND salário\_emp > 25000;

/\* Liste o código dos empregados que trabalham para o departamento 5 ou que supervisionam um empregado que trabalha para o departamento 5 \*/

(SELECT cod\_empregado

FROM empregado

WHERE nro\_departamento = 5)

UNION

(SELECT cod\_supervisor

FROM empregado

WHERE nro\_departamento = 5)

/\* Recupere, para cada empregado do sexo feminino, o seu nome completo e os nomes dos seus dependentes. Use a operação de produto cartesiano \*/

SELECT primeiro\_nome\_emp, último\_nome\_emp, nome\_dependente

FROM empregado, dependente

WHERE sexo\_emp = "feminino" AND empregad.cod\_empregado = dependente.cod\_empregado

/\* Recupere, para cada departamento, o seu nome e o nome completo de seu gerente \*/

SELECT nome\_depto, primeiro\_nome\_emp, último\_nome\_emp

FROM departamento, empregado

WHERE departamento.cod\_gerente = empregado.cod\_empregado

/\* Recupere os nomes completos dos empregados que não têm dependentes. \*/

(SELECT primeiro\_nome\_emp, último\_nome\_emp

FROM empregado)

MINUS

(SELECT primeiro\_nome\_emp, último\_nome\_emp

FROM empregado, dependente

WHERE empregado.cod\_empregado = dependente.cod\_empregado)

/\* Recupere, para cada empregado do sexo feminino, o seu nome completo e os nomes dos seus dependentes. Use a operação de junção natural \*/

SELECT primeiro\_nome\_emp, último\_nome\_emp, nome\_dependente

FROM empregado, dependente

WHERE empregado.cod\_empregrado = dependente.cod\_empregado

AND empregado.sexo\_emp = "feminino"

/\* Para cada projeto localizado no Bloco 19, liste: (i) o número do projeto; (ii) o nome do departamento que controla o projeto; e (iii) o nome completo, o endereço e a data de aniversário do gerente do projeto \*/

SELECT nro\_projeto, nome\_depto, primeiro\_nome\_emp, ultimo\_nome\_emp, end\_emp, data\_niver\_emp

FROM projeto, departamento, empregado

WHERE projeto.nro\_departamento = departamento.nro\_departamento AND

departamento.cod\_gerente = empregado.cod\_empregado AND

projeto.local\_projeto = "Bloco 19"

/\* Recupere o nome completo e o endereço de todos os empregados que trabalham para o Departamento de Informática \*/

SELECT primeiro\_nome\_emp, último\_nome\_emp, end\_emp

FROM empregado, departamento

WHERE nome\_depto = "informatica" AND

departamento.nro\_departamento = empregado.nro\_departamento

/\* Encontre os nomes dos empregados que trabalham em todos os projetos controlados pelo departamento número 5 \*/

SELECT primeiro\_nome\_emp, último\_nome\_emp

FROM empregado

WHERE cod\_empregado IN (

SELECT cod\_empregado

FROM projeto, trabalha\_para

WHERE projeto.nro\_departamento = 5 AND

trabalha\_para.nro\_projeto = projeto.nro\_projeto)

/\* Liste os números dos projetos nos quais que existe um empregado cujo último nome é Silva que trabalha no projeto tanto como um funcionário quanto como um gerente do departamento que controla o projeto \*/

(SELECT DISTINCT nro\_projeto

FROM trabalha\_para T, empregado E

WHERE T.cod\_empregado = E.cod\_empregado AND

último\_nome\_emp = “Silva” )

UNION

(SELECT DISTINCT nro\_projeto

FROM projeto P, departamento D, empregado E

WHERE P.nro\_departamento = D.nro\_departamento AND

E.cod\_empregado = D.cod\_gerente AND

E.último\_nome\_emp = “Silva”)

/\* Liste os nomes completos dos gerentes que tem pelo menos um dependente \*/

SELECT DISTINCT primeiro\_nome\_emp, último\_nome\_emp

FROM empregado, departamento

WHERE empregado.cod\_empregado = departamento.cod\_gerente AND

cod\_empregado IN

(SELECT DISTINCT cod\_empregado

FROM dependente)

/\* Liste a soma dos salários de todos os empregados que trabalham para o Departamento de Informática, assim como o maior salário, o menor salário e a média dos salários desses empregados \*/

SELECT SUM(salário\_emp), MAX(salário\_emp), MIN(salário\_emp),

AVG(salário\_emp)

FROM empregado, departamento

WHERE empregado.nro\_departamento = departamento.nro\_departamento AND

nome\_depto = “Departamento de Informática”

/\* Recupere o número total de empregados que trabalham para o Departamento de Informática. \*/

SELECT COUNT (\*)

FROM empregado, departamento

WHERE empregado.nro\_departamento = departamento.nro\_departamento AND

nome\_depto = “Departamento de Informática”

/\* Para cada projeto, liste o número do projeto, o nome do projeto e o número de empregados que trabalham para o projeto \*/

SELECT nro\_projeto, nome\_projeto, COUNT(\*)

FROM projeto, trabalha\_para

WHERE projeto.nro\_projeto = trabalha\_para.nro\_projeto

GROUP BY nro\_projeto, nome\_projeto