MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

SQL - Structured Query Language

Karine Reis Ferreira

karine@dpi.inpe.br

SQL - Structured Query Language

- Linguagem padrão (ISO) para sistemas de bancos de dados
- É uma linguagem declarativa de alto nível que permite:
  - Consultar dados
  - Definir e alterar dados
  - Definir visões
  - Especificar autorização e regras de segurança
  - Definir restrições de integridade
  - Criar índices
  - Controlar transações
  - ...

SQL - Structured Query Language

- Linguagem de consulta usada pelos SGBD-R e SGBD-OR
- Baseada na álgebra relacional
- É divida em:
  - Linguagem de manipulação de dados (SQL DML)
  - Linguagem de definição de dados (SQL DDL)
  - Definição de visões (SQL DDL)
  - Especificação de autorização (SQL DDL)
  - Especificação de integridade (SQL DDL)
  - Controle de transação (SQL DDL)

SQL - Structured Query Language

SQL-DDL

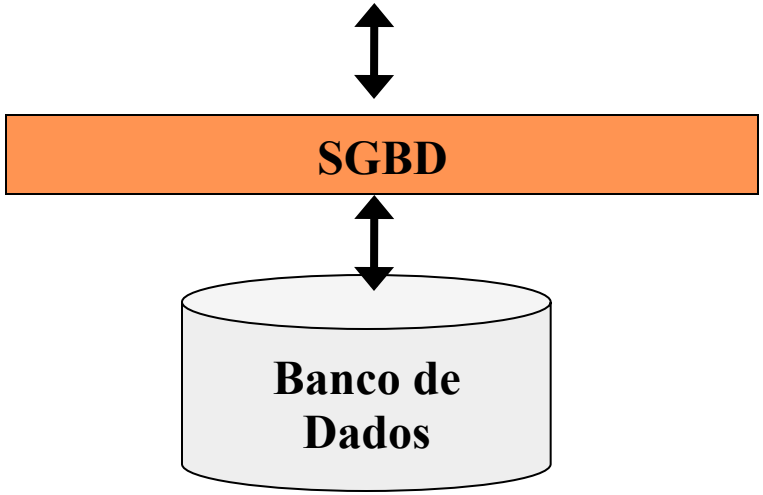
```
CREATE DATABSE Teste
```

```
CREATE TABLE Estados (
NOME     VARCHAR(100)
SIGLA    VARCHAR(2)
POP      NUMBER(10,10))
```

SQL-DML

```
INSERT INTO Estados
VALUES ("Minas
Gerais", "MG", 9999)
```

```
SELECT *
FROM   Estados
WHERE  SIGLA = "MG"
```



SQL - Structured Query Language

Alguns comandos SQL:

Comandos	Usado para	Tipo
select	Consultar dados	DML
insert, *update*, *delete*	Incluir, alterar e remover dados	DML
commit, *rollback*	Controlar transações	DDL
create, *alter*, *drop*	Definir, alterar e remover esquemas (tabelas)	DDL

SQL - Structured Query Language

DDL– Data Definition Language

CREATE DATABASE – cria um novo banco de dados

ALTER DATABASE – modifica um banco de dados

CREATE TABLE – cria uma nova tabela

ALTER TABLE – altera uma tabela

DROP TABLE – remove uma tabela

CREATE INDEX – cria um índice

DROP INDEX – remove um índice

SQL - Structured Query Language

DML – Data Manipulation Language

SELECT – extrai dados de um banco de dados

UPDATE – altera os dados de um banco de dados

DELETE – apaga dados de um banco de dados

INSERT INTO – insere dados no banco de dados

SQL – Create database – Example

```
CREATE DATABASE lab_bdgeo
WITH OWNER = postgres
ENCODING = 'UTF8'
TABLESPACE = pg_defaultt;
```

SQL – Create table

```
CREATE TABLE <table name>
(  <column name> <column type> [<attribute constraint>]
{, <column name> <column type> [<attribute constraint>] }
[<table constraint> {, <table constraint> } ] )
```

[…]: opcional

{…}: repetições -> 0 or *n* vezes

| : mutualmente exclusivos

SQL – Column type

Numeric	Integer numbers	`INT` ou `INTEGER`, `SMALLINT`
	Floating-point numbers	`FLOAT` ou `REAL`, `DOUBLE PRECISION`
	Formated numbers: *i* (precision): number of decimal digits and *j* (scale): number of digits after decimal point	`DECIMAL (i,j)` ou `DEC(i,j)` ou `NUMERIC(i,j)`
Character-string	Fixed length with *n* characters	`CHAR(n)` ou `CHARACTER(n)`
	Varying length with maximum *n* characters	`VARCHAR(n)` ou `CHAR VARYING(n)` ou `CHARACTER VARYING(n)`
	Large text values (ex. documents)	`CHARACTER LARGE OBJECT (CLOB)`
Bit-string	Fixed length with *n* bits	`BIT(n)`
	Varying length with maximum *n* bits	`BIT VARYING(n)`
	Large binary values (ex. images)	`BIT LARGE OBJECT (BLOB)`

SQL – Column type

Boolean	Values of TRUE or FALSE or UNKNOWN	`BOOLEAN`
Date	YEAR, MONTH, and DAY (YYYY-MM-DD)	`DATE`
Time	HOUR, MINUTE, and SECOND (HH:MM:SS) with or without time zone (HOURS:MINUTES)	`TIME` e `TIME WITH TIME ZONE`
Timestamp	Both date and time, with or without time zone	`TIMESTAMP` e `TIMESTAMP WITH TIME ZONE`
Time interval	A relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp.	`INTERVAL`

SQL – Constraints

Restringir que um atributo não tenha valores nulos	`NOT NULL`
Restringir valores e domínios de atributos	`CHECK (<expression>)`
Restringir que um ou mais atributos tenham valores únicos	`UNIQUE (<column_name> {,<column_name>})`
Definir chave primária	`PRIMARY KEY (<column_name> {,<column_name>})`

Definir restrições de integridade referencial (chave estrangeira)	`FOREIGN KEY (<column_name> {,<column_name>})`
	`REFERECES <table_name>`
	`(<column_name> {,<column_name>})`
	`ON DELETE (SET DEFAULT | SET NULL | CASCADE)`
	`ON UPDATE (SET DEFAULT | SET NULL | CASCADE)`

SQL – Create table – Examples

```
CREATE TABLE Cliente (
    ClienteID   INT            NOT NULL,
    Empresa     VARCHAR( 40 )  NOT NULL,
    Endereco    VARCHAR( 60 ),
    Cidade      VARCHAR( 15 ),
    Estado      VARCHAR( 15 ),
    CEP         VARCHAR( 10 ),
    Pais        VARCHAR( 15 ),
    Fone        VARCHAR( 24 ),
    CONSTRAINT ClientePK PRIMARY KEY (ClienteID)
);


CREATE TABLE Fornecedor (
    FornecedorID    INT             NOT NULL,
    Nome            VARCHAR( 40 )   NOT NULL,
    Endereco        VARCHAR( 60 ),
    CONSTRAINT FornecedorPK PRIMARY KEY (FornecedorID)
);

CREATE TABLE Produto (
    ProdutoID       INT             NOT NULL,
    FornecedorID    INT             NOT NULL,
    Descricao       VARCHAR( 40 )   NOT NULL,
    PrecoUnitario   REAL            NOT NULL  CHECK(PrecoUnitario>=0),

    CONSTRAINT ProdutoPK        PRIMARY KEY (ProdutoID),

    CONSTRAINT FornecProdutoFK  FOREIGN KEY (FornecedorID) REFERENCES Fornecedor(FornecedorID)
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
);
```

SQL – Create table – Examples

```
CREATE TABLE Vendedor (
    VendedorID  INT             NOT NULL,
    Nome        VARCHAR( 40 )   NOT NULL,

    PRIMARY KEY ( VendedorID )
);

CREATE TABLE Pedido (
    PedidoID    INT             NOT NULL,
    VendedorID  INT             NOT NULL,
    ClienteID   INT             NOT NULL,
    ProdutoID   INT             NOT NULL,
    Quantidade  INT             NOT NULL    DEFAULT 1  CHECK(Quantidade>0),

    PRIMARY KEY ( PedidoID ),

    CONSTRAINT PedidoVendedorFK FOREIGN KEY (VendedorID) REFERENCES Vendedor(VendedorID)
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,

    CONSTRAINT PedidoClienteFK FOREIGN KEY (ClienteID) REFERENCES Cliente(ClienteID)
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,

    CONSTRAINT PedidoProdutoFK FOREIGN KEY (ProdutoID) REFERENCES Produto(ProdutoID)
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
);
```

SQL – Create table – Examples

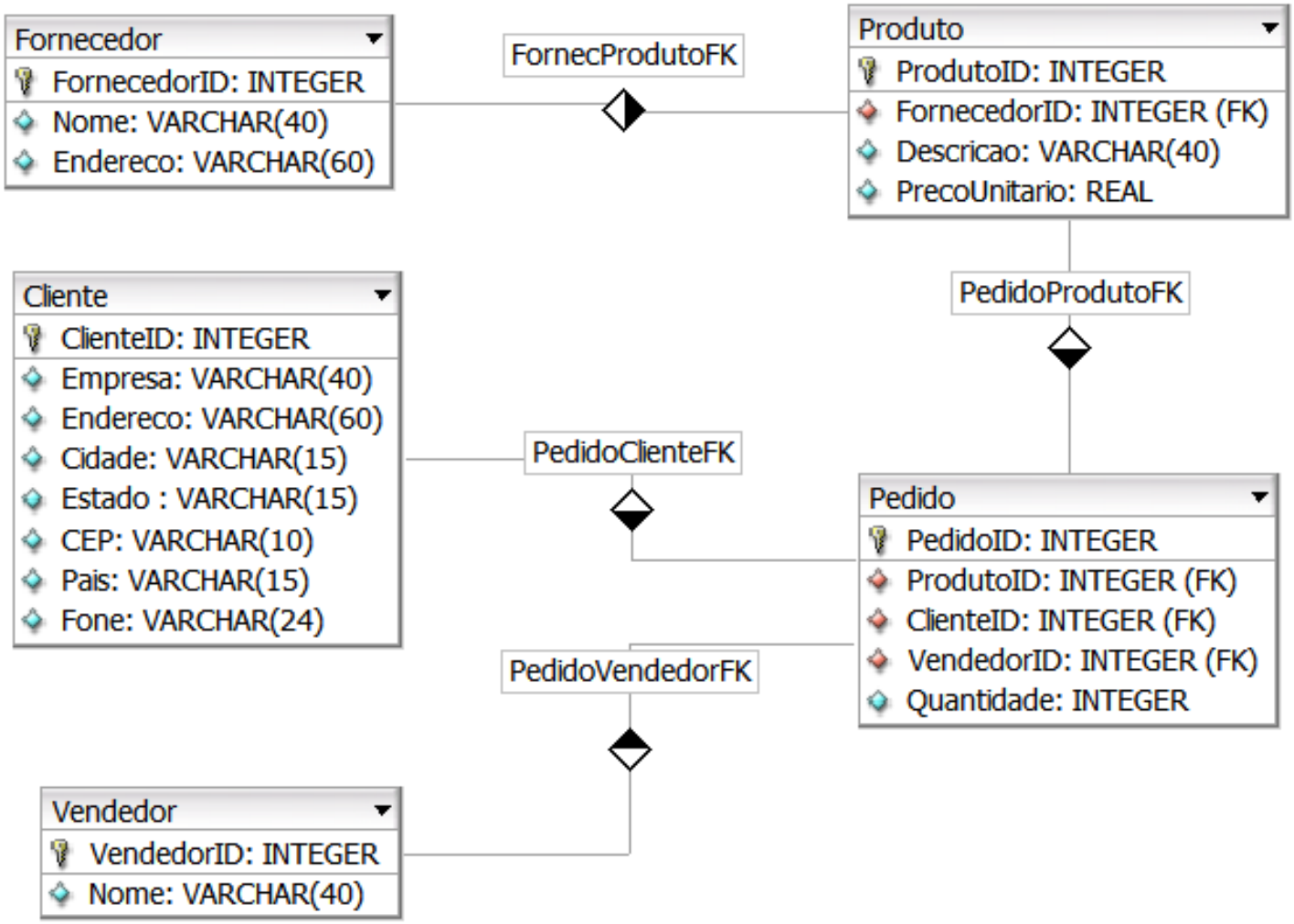
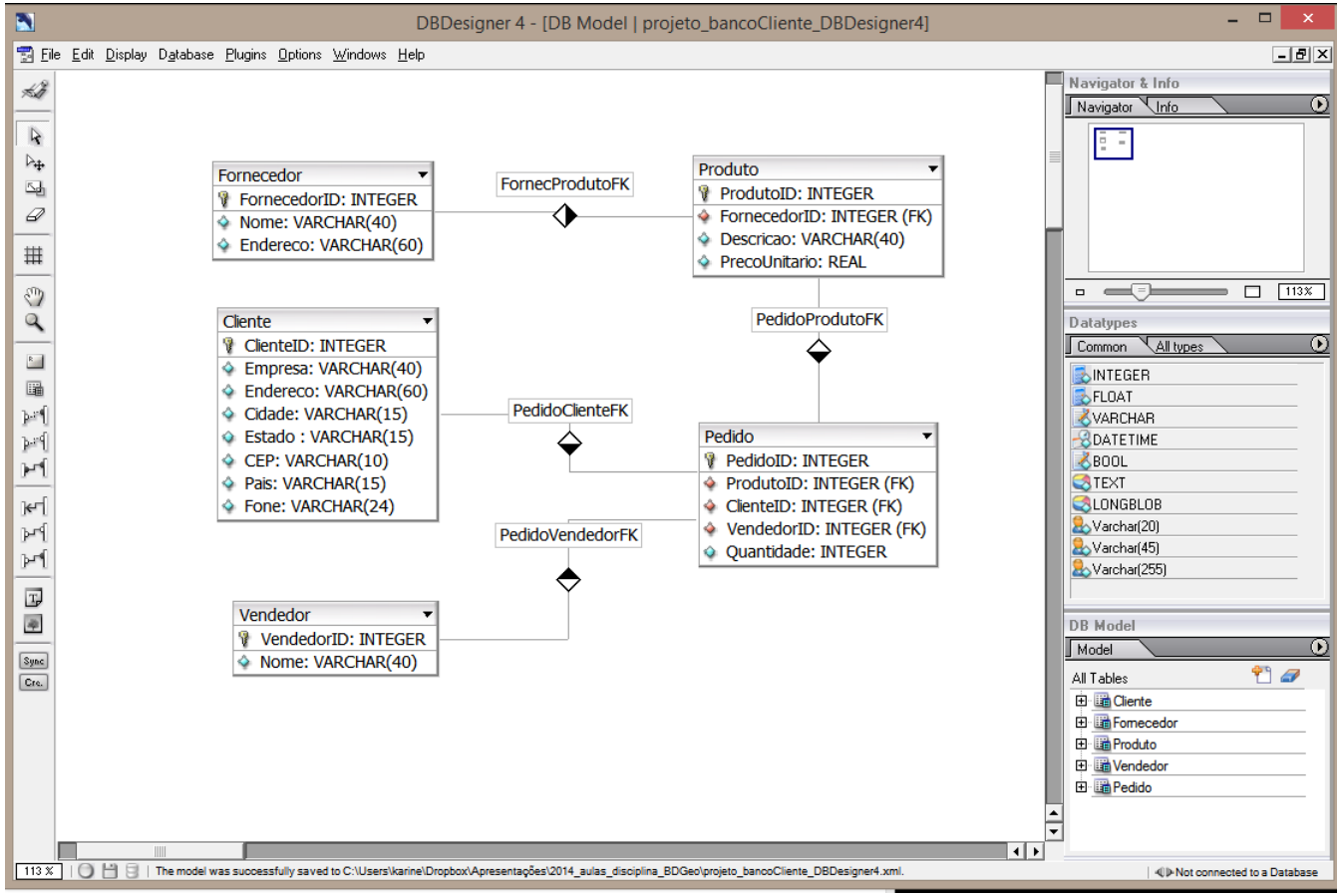


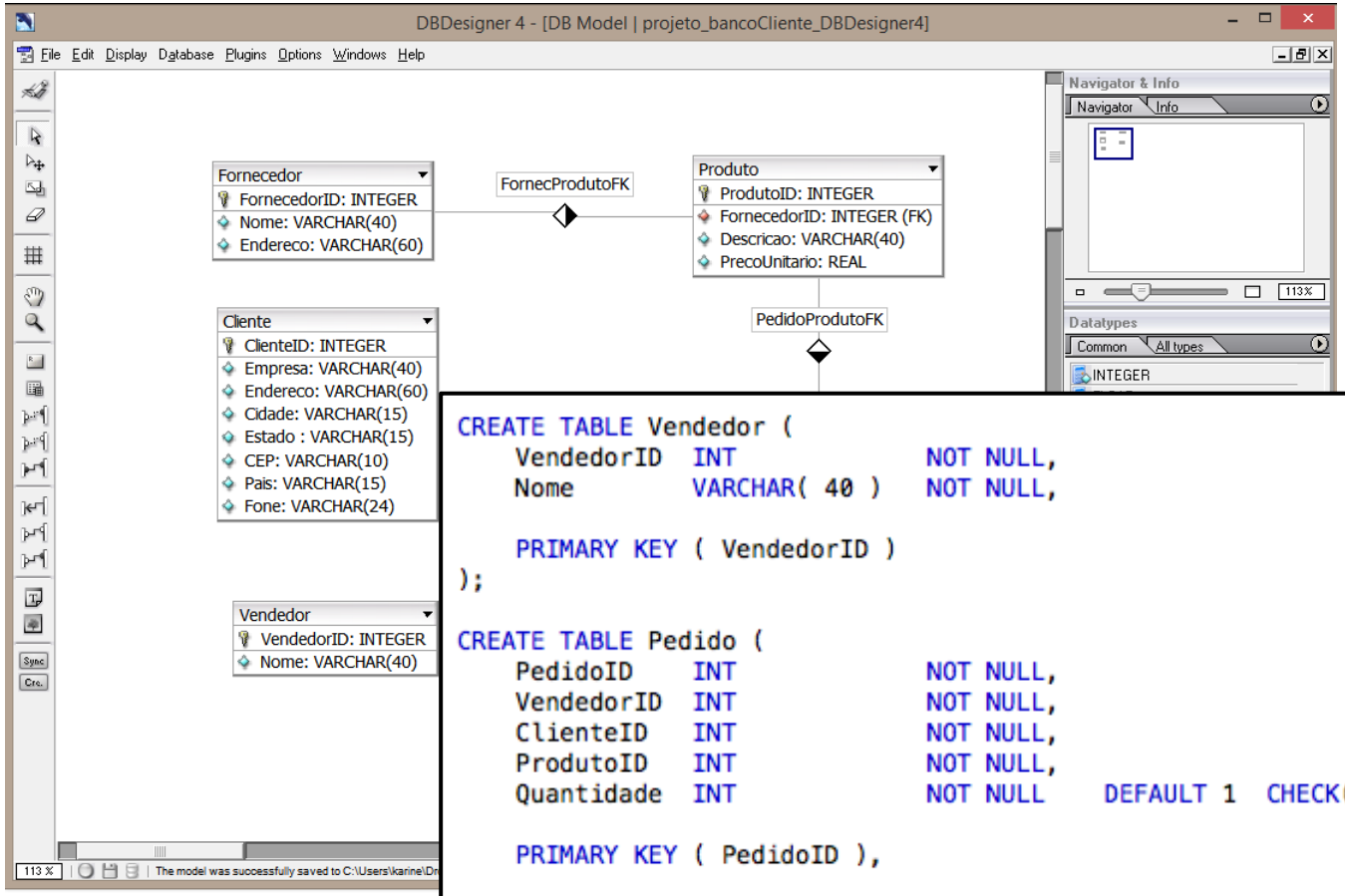
Diagrama criado com o aplicativo DBDesigner 4.

DBDesigner 4

- ✓ DBDesigner 4 is a visual database design system that integrates database design, modeling, creation and maintenance into a single, seamless environment.
- ✓ Open Source (GPL)
- ✓ http://www.fabforce.net/dbdesigner4/
- ✓ Developed and optimized for the open source MySQL-Database, but it can create standard SQL scripts from its diagrams


DBDesigner 4 - [DB Model | projeto_bancoCliente_DBDesigner4]
File Edit Display Database Plugins Options Windows Help
Fornecedor
FornecedorID: INTEGER
Nome: VARCHAR(40)
Endereco: VARCHAR(60)
FornecProdutoFK
Produto
ProdutoID: INTEGER
FornecedorID: INTEGER (FK)
Descricao: VARCHAR(40)
PrecoUnitario: REAL
PedidoProdutoFK
Cliente
ClienteID: INTEGER
Empresa: VARCHAR(40)
Endereco: VARCHAR(60)
Cidade: VARCHAR(15)
Estado : VARCHAR(15)
CEP: VARCHAR(10)
Pais: VARCHAR(15)
Fone: VARCHAR(24)
PedidoClienteFK
Pedido
PedidoID: INTEGER
ProdutoID: INTEGER (FK)
ClienteID: INTEGER (FK)
VendedorID: INTEGER (FK)
Quantidade: INTEGER
PedidoVendedorFK
Vendedor
VendedorID: INTEGER
Nome: VARCHAR(40)
Navigator & Info
Navigator
Info
Datatypes
Common
All types
INTEGER
FLOAT
VARCHAR
DATETIME
BOOL
TEXT
LONGBLOB
Varchar(20)
Varchar(45)
Varchar(255)
DB Model
Model
All Tables
Cliente
Fornecedor
Produto
Vendedor
Pedido
The model was successfully saved to C:\Users\karine\Dropbox\Apresentações\2014_aulas_disciplina_BDGeo\projeto_bancoCliente_DBDesigner4.xml.
Not connected to a Database


THE MYSQL DATA-
BASE DESIGNER
DBDesigner 4




```
CREATE TABLE Vendedor (
    VendedorID  INT             NOT NULL,
    Nome        VARCHAR( 40 )   NOT NULL,

    PRIMARY KEY ( VendedorID )
);

CREATE TABLE Pedido (
    PedidoID    INT             NOT NULL,
    VendedorID  INT             NOT NULL,
    ClienteID   INT             NOT NULL,
    ProdutoID   INT             NOT NULL,
    Quantidade  INT             NOT NULL    DEFAULT 1  CHECK(Quantidade>0),

    PRIMARY KEY ( PedidoID ),

    CONSTRAINT PedidoVendedorFK FOREIGN KEY (VendedorID) REFERENCES Vendedor(VendedorID)
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,

    CONSTRAINT PedidoClienteFK FOREIGN KEY (ClienteID) REFERENCES Cliente(ClienteID)
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,

    CONSTRAINT PedidoProdutoFK FOREIGN KEY (ProdutoID) REFERENCES Produto(ProdutoID)
                                ON DELETE CASCADE
                                ON UPDATE CASCADE
);
```

SQL scripts from diagrams

SQL – Insert table

```
INSERT INTO <table name>
[ ( <column name> {, <column name> } ) ]
( VALUES ( <constant value>, { <constant value> } )
{, ( <constant value> {, <constant value> } ) }
| <select statement> )
```

[…]: opcional

{…}: repetições -> 0 or *n* vezes

| : mutualmente exclusivos

SQL – Insert table – Examples

```
INSERT INTO Cliente Values ( 1, 'ACM', 'Rua das Flores, 10', 'Sao Paulo', 'SP', '1222000', 'Brasil', '112233445566');
INSERT INTO Cliente Values ( 2, 'VW', 'Rua do Comercio, 47', 'Sao Paulo', 'SP', '1222010', 'Brasil', '11298735566');
INSERT INTO Cliente Values ( 3, 'GM', 'Via Dutra, 1000', 'Sao Jose dos Campos', 'SP', '1222560', 'Brasil', '122239876566');
INSERT INTO Cliente Values ( 4, 'TEX', 'AV Brasil, 1210', 'Rio de Janeiro', 'RJ', '348890', 'Brasil', '212134567');

INSERT INTO Vendedor Values ( 1, 'Jose Marcio');
INSERT INTO Vendedor Values ( 2, 'Luis Claudio');
INSERT INTO Vendedor Values ( 3, 'Andre Carlos');

INSERT INTO Fornecedor Values ( 1, 'Ferragens Santa Lucia', 'Rua Catalao, 20, Goiania, GO');
INSERT INTO Fornecedor Values ( 2, 'Borracharia Campos', 'Rua dos Ipes 1235, Presidente Prudente, SP');
INSERT INTO Fornecedor Values ( 3, 'Tintas Brasil', 'Avenida dos Guararapes 44, Paulinia, SP');

INSERT INTO Produto Values ( 1, 2, 'Roda', 500.00);
INSERT INTO Produto Values ( 2, 1, 'Mola', 234.00);
INSERT INTO Produto Values ( 3, 1, 'Porca', 11.00);
INSERT INTO Produto Values ( 4, 1, 'Parafuso', 5.30);
INSERT INTO Produto Values ( 5, 2, 'Prego', 1.20);
INSERT INTO Produto Values ( 6, 3, 'Tinta', 234.00);

INSERT INTO Pedido Values ( 1, 2, 4, 2, 450);
INSERT INTO Pedido Values ( 2, 1, 2, 1, 123);
INSERT INTO Pedido Values ( 3, 2, 1, 2, 60);
INSERT INTO Pedido Values ( 4, 3, 2, 2, 121);
INSERT INTO Pedido Values ( 5, 3, 3, 6, 65);
INSERT INTO Pedido Values ( 6, 1, 3, 5, 36);
INSERT INTO Pedido Values ( 7, 2, 1, 5, 140);
INSERT INTO Pedido Values ( 8, 3, 4, 1, 200);
INSERT INTO Pedido Values ( 9, 3, 2, 3, 67);
INSERT INTO Pedido Values ( 10, 1, 2, 3, 89);
```

SQL – Insert table – Examples

```
CREATE TABLE Cliente2 (
    ClienteID   INT             NOT NULL,
    Empresa     VARCHAR( 40 )   NOT NULL,
    Endereco    VARCHAR( 60 ),
    Cidade      VARCHAR( 50 ),
    Estado      VARCHAR( 15 ),
    CEP         VARCHAR( 10 ),
    Pais        VARCHAR( 15 ),
    Fone        VARCHAR( 24 ),
    CONSTRAINT Cliente2PK PRIMARY KEY (ClienteID)
);


INSERT INTO Cliente2 (SELECT * FROM Cliente);
```

SQL – Catalog

- SQL2 uses the concept of a catalog: a named collection of schemas in an SQL environment.

- Schema INFORMATION_SCHEMA: information on all the schemas in the catalog and all the element descriptors in these schemas.

- Tables:

CHECK_CONSTRAINTS	TABLES
COLUMNS	TABLE_CONSTRAINTS
COLUMN_PRIVILEGES	TABLE_PRIVILEGES
CONTRAINT_COLUMN_USAGE	USER_DEFINED_TYPES
CONSTRAINT_TABLE_USAGE	VIEWS
REFERENTIAL_CONSTRAINT	…

SQL – Catalog – Examples

```
SELECT * FROM information_schema.tables

SELECT * FROM information_schema.columns

SELECT * FROM information_schema.views

SELECT * FROM information_schema.triggers

...
```

SQL – Select-From-Where

```
SELECT [ DISTINCT ] <attribute list>
FROM   (<table name> {<alias>} | <joined table> )
       {(<table name> {<alias>} | <joined table>)}
[ WHERE <condition> ]
[ GROUP BY   <grouping attributes>
             [HAVING <group selection condition>] ]
[ ORDER BY   <column name> [<order>]
             {, <column name> [<order>] }]
```

[…]: opcional

{…}: repetições -> 0 or *n* vezes

| : mutualmente exclusivos

SQL – Select-From-Where

```
SELECT [ DISTINCT ] <attribute list>

<attribute list> := ...
```

Define quais colunas farão parte do resultado da consulta

Equivale ao operador *projeção* da álgebra relacional

Opções	Descrição
`DISTINCT`	Indica que as linhas duplicadas devem ser eliminadas do resultado
*	Indica que todas as colunas de todas as tabelas da cláusula FROM devem ser incluídas no resultado
`<column_name>`	Nome de uma coluna de uma tabela da cláusula FROM que será incluída no resultado.
`<function>`	Funções definidas em SQL como, por exemplo, funções de agregação (ex.: *avg*, *min*, *max*, *count*, etc)

SQL – Select-From-Where

```
FROM   (<table name> {<alias>}
                | <joined table> )
       {(<table name> {<alias>}
                | <joined table>)}
```

Define quais tabelas serão consultadas

Equivale ao operador *produto cartesiano* da álgebra relacional

Opções	Descrição
<alias>	Nome alternativo para uma coluna, expressão ou tabela
<table_name>	Nome de uma tabela envolvida na consulta
<jointed_table>	Junção de tabelas envolvidas na consulta

SQL – Jointed Table

```
SELECT *
FROM table1 INNER JOIN table2 ON table1.id = table2.id
```

```
SELECT *
FROM table1 LEFT JOIN table2 ON table1.id = table2.id
```

```
SELECT *
FROM table1 RIGHT JOIN table2 ON table1.id = table2.id
```

SQL – Select-From-Where

```
[ WHERE <condition> ]
```

Define quais as restrições que as linhas das tabelas da cláusula FROM devem satisfazer para entrarem no resultado

Equivale ao operador *seleção* da álgebra relacional

Opções	Descrição
`<condition>`	Uma condição à qual as linhas das tabelas da cláusula FROM devem satisfazer para entrarem no resultado

SQL – Select-From-Where

```
[ GROUP BY <grouping attributes>
        [HAVING
        <group selection condition>] ]
```

GROUP BY: Indica que o resultado deve ser agrupado

HAVING: Indica quais os grupos gerados pela cláusula GROUP BY entrarão no resultado

Opções	Descrição
<column_name>	Uma ou mais colunas cujos valores serão usados para agrupar o resultado.

Opções	Descrição
<group_selection_condition>	Uma condição à qual os grupos gerados pela cláusula GROUP BY devem satisfazer para entrarem no resultado.

SQL – Select-From-Where

```
[ ORDER BY    <column name> [<order>]
              {, <column name>
                    [<order>] }]
```

Indica como o resultado deve ser ordenado

Opções	Descrição
<column_name> [(ASC | DESC)]	Uma ou mais colunas cujos valores serão usados para ordenar o resultado. A ordenação pode ser de forma ASCENDENTE ou DESCENDENTE.

SQL – Operadores de Agregação

- AVG(…): média dos valores da coluna
- SUM(…): soma dos valores da coluna
- COUNT(…): número de valores na coluna
- MAX(…): maior valor na coluna
- MIN(…): menor valor na coluna
- ...

Podem ser aplicados pra todos os registros de uma coluna ou para grupos de registros (usando a cláusula GROUP BY)

SQL – Select-From-Where – Examples

- ✓ Selecione todos os clientes ordenados pela empresa
- ✓ Selecione as empresas, enderecos e telefones de todos os clientes, ordenado pelo endereco
- ✓ Quantos clientes existem?

```
SELECT * FROM cliente ORDER BY empresa;
```

```
SELECT empresa, endereco, fone FROM cliente
ORDER BY endereco;
```

```
SELECT COUNT (*) FROM cliente;
```

SQL – Select-From-Where – Examples

- ✓ Selecione todos os pedidos do cliente "ACM"
- ✓ Quantos itens o cliente "ACM" comprou?

```
SELECT *
FROM cliente INNER JOIN pedido
     ON cliente.clienteid = pedido.clienteid
WHERE cliente.empresa = 'ACM';


SELECT SUM (pedido.quantidade)
FROM cliente INNER JOIN pedido
     ON cliente.clienteid = pedido.clienteid
WHERE cliente.empresa = 'ACM';
```

SQL – Select-From-Where – Examples

- ✓ Quantos itens cada cliente comprou?
- ✓ Quais clientes compraram mais que 200 itens?

```
SELECT cliente.empresa, SUM (pedido.quantidade)
FROM cliente INNER JOIN pedido
     ON cliente.clienteid = pedido.clienteid
GROUP BY cliente.empresa;


SELECT cliente.empresa, SUM (pedido.quantidade)
FROM cliente INNER JOIN pedido
     ON cliente.clienteid = pedido.clienteid
GROUP BY cliente.empresa
HAVING SUM (pedido.quantidade) > 200
```

SQL – Select-From-Where – Examples

- ✓ Selecione todas as informações dos pedidos: identificador do pedido, nome do vendedor, descricao do produto, nome do fornecedor e quantidade comprada.

```
SELECT  pedido.pedidoid as pedidoid, vendedor.nome as vendedor,
        cliente.empresa as cliente, produto.descricao as produto,
        fornecedor.nome as fornecedor, pedido.quantidade
FROM (((pedido INNER JOIN vendedor ON pedido.vendedorid = vendedor.vendedorid)
     INNER JOIN cliente ON pedido.clienteid = cliente.clienteid)
     INNER JOIN produto ON pedido.produtoid = produto.produtoid)
     INNER JOIN fornecedor ON produto.fornecedorid = fornecedor.fornecedorid
ORDER BY cliente
```

SQL – Create view

Tabela Virtual ou *View* é uma tabela que é derivada de outras tabelas e não existe fisicamente armazenada no banco de dados.

```
CREATE VIEW <view_name>
[(<column_name> {, <column_name> })]
AS <select statement>
```

SQL – Create view – Examples

```
CREATE VIEW pedido_descricao (pedidoid, vendedor, empresa, produto, fornecedor, quantidade)
AS
SELECT  pedido.pedidoid, vendedor.nome, cliente.empresa, produto.descricao,
        fornecedor, pedido.quantidade
FROM    (((pedido INNER JOIN vendedor ON pedido.vendedorid = vendedor.vendedorid)
        INNER JOIN cliente ON pedido.clienteid = cliente.clienteid)
        INNER JOIN produto ON pedido.produtoid = produto.produtoid)
        INNER JOIN fornecedor ON produto.fornecedorid = fornecedor.fornecedorid



SELECT * FROM pedido_descricao
```

SQL – Update

- ✓ Altera valores dos registros das tabelas

```
UPDATE <table name>
SET <column name> = <new value>
{, <column name> = <new value>}
[ WHERE <condition> ]
```

SQL – Update – Examples

```
UPDATE cliente
SET endereco = 'Rua das Flores, 505'
WHERE empresa = 'ACM'


UPDATE pedido
SET quantidade = quantidade * 2
```

SQL – Delete

- Remove registros das tabelas

```
DELETE <table name>
[ WHERE <condition> ]
```

SQL – Delete – Examples

```
DELETE FROM vendedor
WHERE nome = 'Andre Carlos'

DELETE FROM vendedor
```

OBS 1: Note que após executar o primeiro comando, todos os pedidos associados ao vendedor “Andre Carlos” são removidos da tabela “Pedido”. Isso acontece porque a restrição entre as tabelas “Vendedor” e “Pedido” foi criada com a ação “ON DELETE CASCADE”!

OBS 2: Note que após executar o segundo comando, todos os vendedores e pedidos são removidos do banco. Isso acontece porque a restrição entre as tabelas “Vendedor” e “Pedido” foi criada com a ação “ON DELETE CASCADE”!

SQL – Alter table

```
ALTER TABLE <table name> ADD <column definition>
ALTER TABLE <table name> ADD COLUMN <column definition>

ALTER TABLE <table name> DROP COLUMN <column name> <action>

ALTER TABLE <table name>   ALTER COLUMN <column name>
                           <new column definition>
ALTER TABLE <table name>   ALTER COLUMN <column name>
                           TYPE <new column type>
```

SQL – Alter table

```
ALTER TABLE <table name> DROP CONSTRAINT <constraint name>
                                         <action>

ALTER TABLE <table name> RENAME COLUMN  <column name> TO
                                        <new column name>

ALTER TABLE <table name> RENAME TO <new table name>
```

SQL – Alter table – Examples

```
ALTER TABLE Cliente ADD CPF VARCHAR( 14 ) NOT NULL DEFAULT 0000000

ALTER TABLE Cliente ADD COLUMN CPF2 VARCHAR( 14 ) NOT NULL DEFAULT 0000000

ALTER TABLE Cliente DROP COLUMN CPF2 CASCADE

ALTER TABLE Cliente ALTER COLUMN CPF DROP NOT NULL

ALTER TABLE Cliente ALTER COLUMN CPF TYPE VARCHAR( 150 )
```

SQL – Drop table

```
DROP TABLE <table name> [(CASCADE | RESTRICT)]
```

- ✓ CASCADE: exclui também todos os objetos relacionados ao objeto excluído
- ✓ RESTRICT: o objeto só é excluído se não há nenhum outro objeto relacionado a ele. (opção default)

SQL – Drop table – Examples

```
DROP TABLE Vendedor

DROP TABLE Vendedor CASCADE
```

OBS: Note que após executar o segundo comando, todas as restrições (*constraints*) relacionadas a essa tabela são removidas.

SQL – Drop table – Examples

```
DROP TABLE vendedor CASCADE;
DROP TABLE fornecedor CASCADE;
DROP TABLE prodduto CASCADE;
DROP TABLE cliente CASCADE;
DROP TABLE pedido CASCADE;
```

OBS: Os comandos acima removem todas as tabelas do banco de dados