Relational Database Systems
– Part 02 –

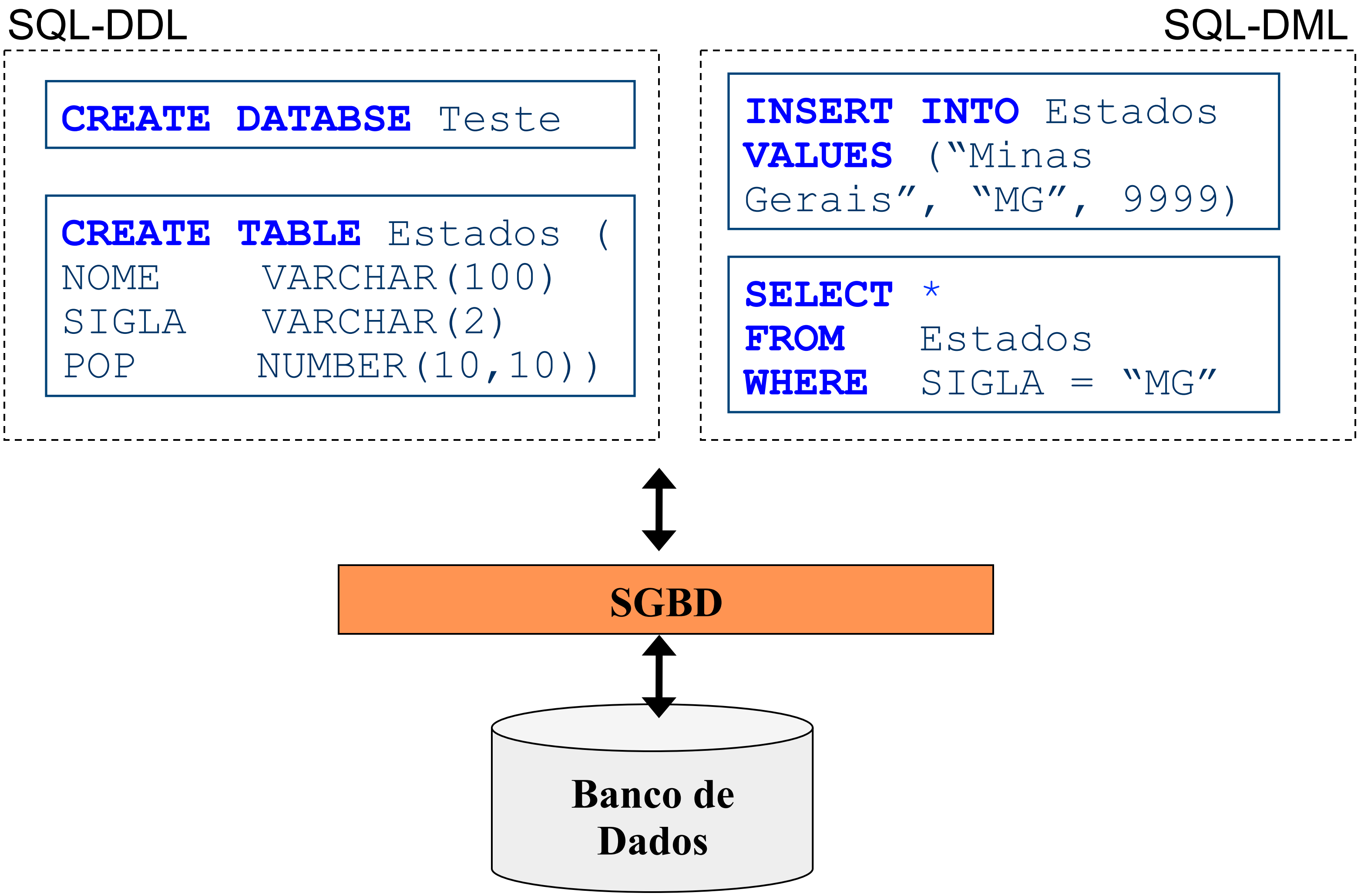
Karine Reis Ferreira

karine.ferreira@inpe.br

SQL: Structured Query Language

- A standard (ISO) for relational databases.
- Based on the *relational algebra*
- *Higher-level declarative language* interface: user only specifies what the result is to be, leaving the actual optimization and decisions on how to execute the query to the DBMS.
- Statements for:
  - ✓ data definitions, queries, and updates: DDL and DML
  - ✓ defining views on the database
  - ✓ specifying security and authorization
  - ✓ defining integrity constraints, and
  - ✓ specifying transaction controls.

SQL: Structured Query Language



From Relational Diagram to SQL Script

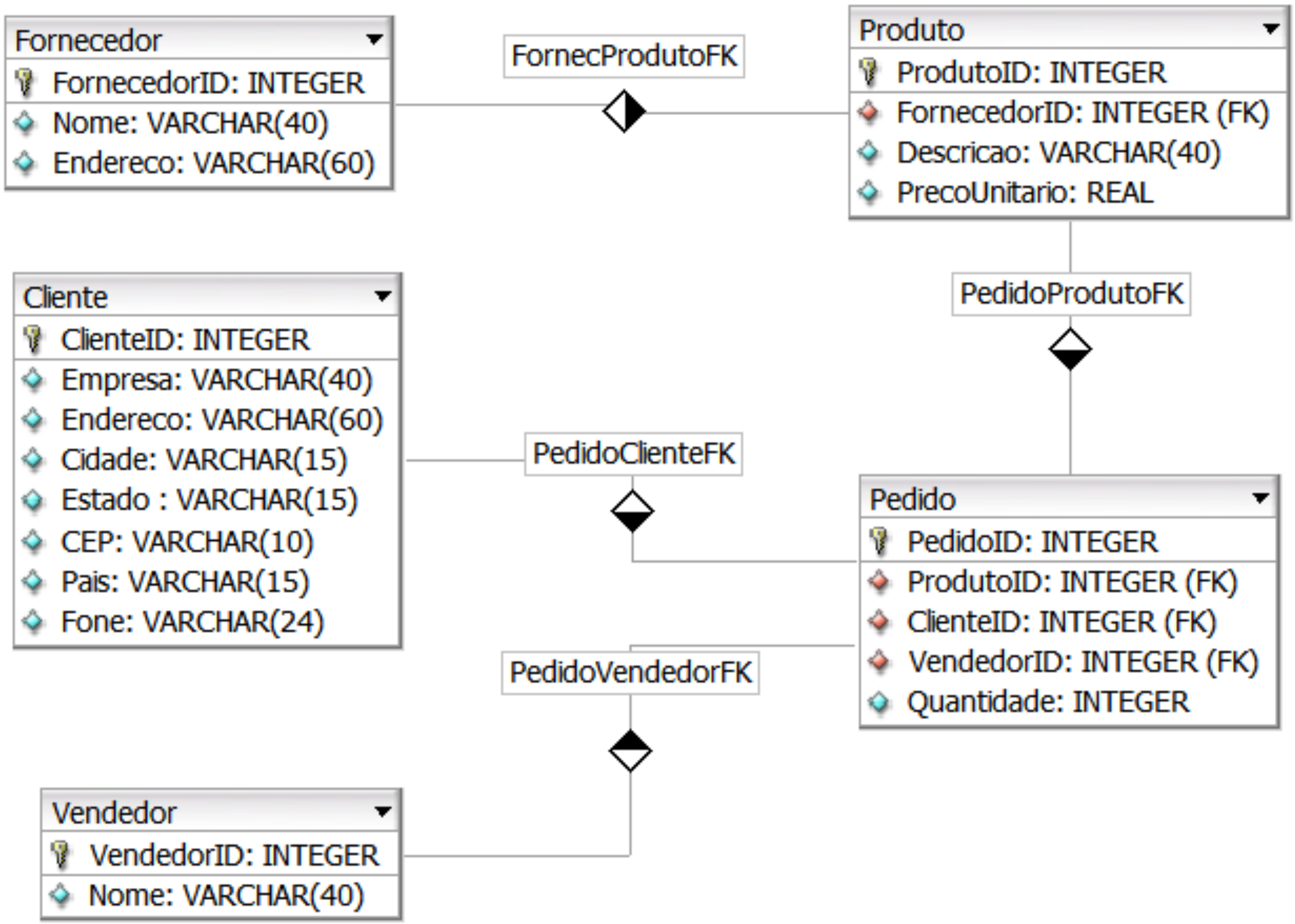
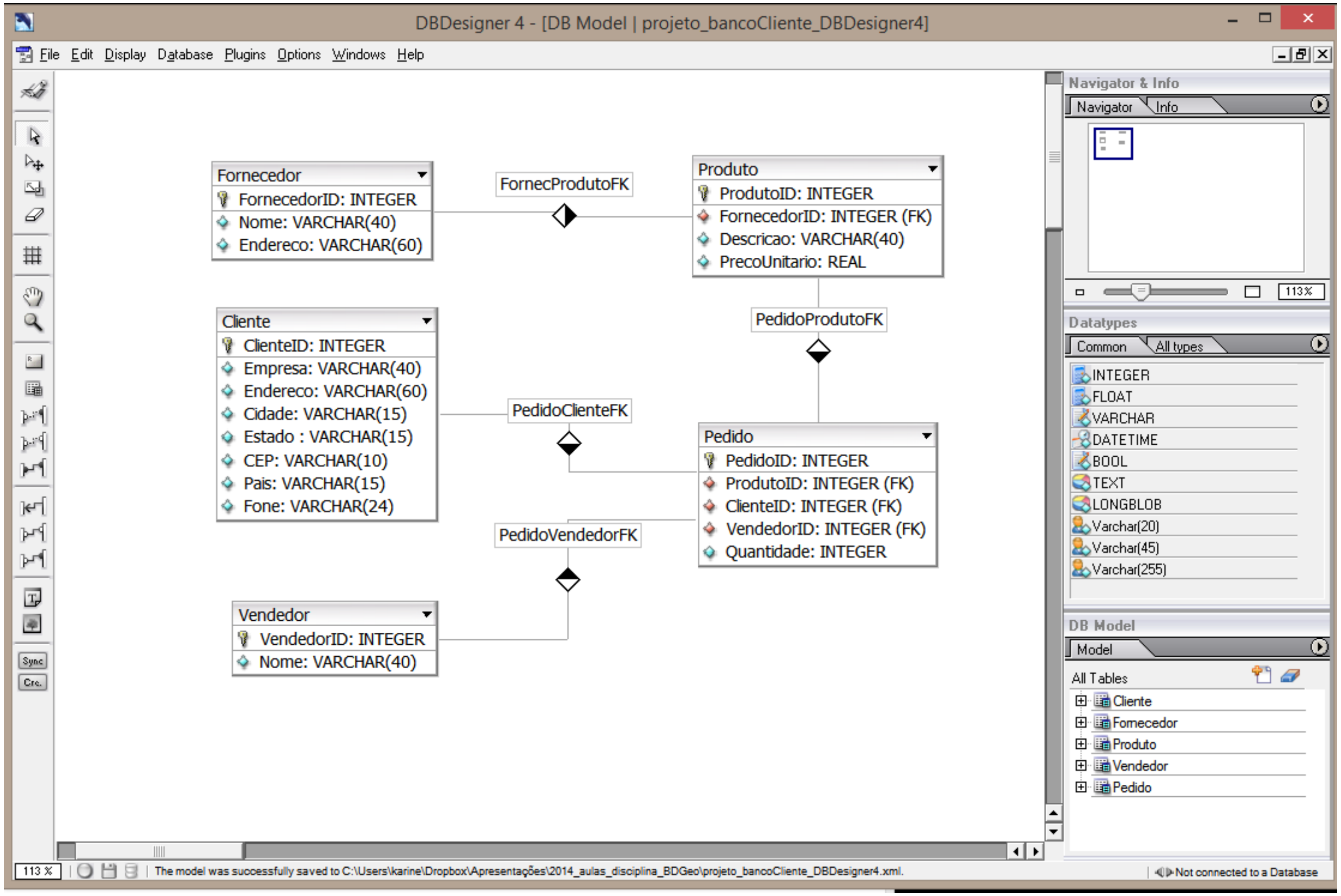


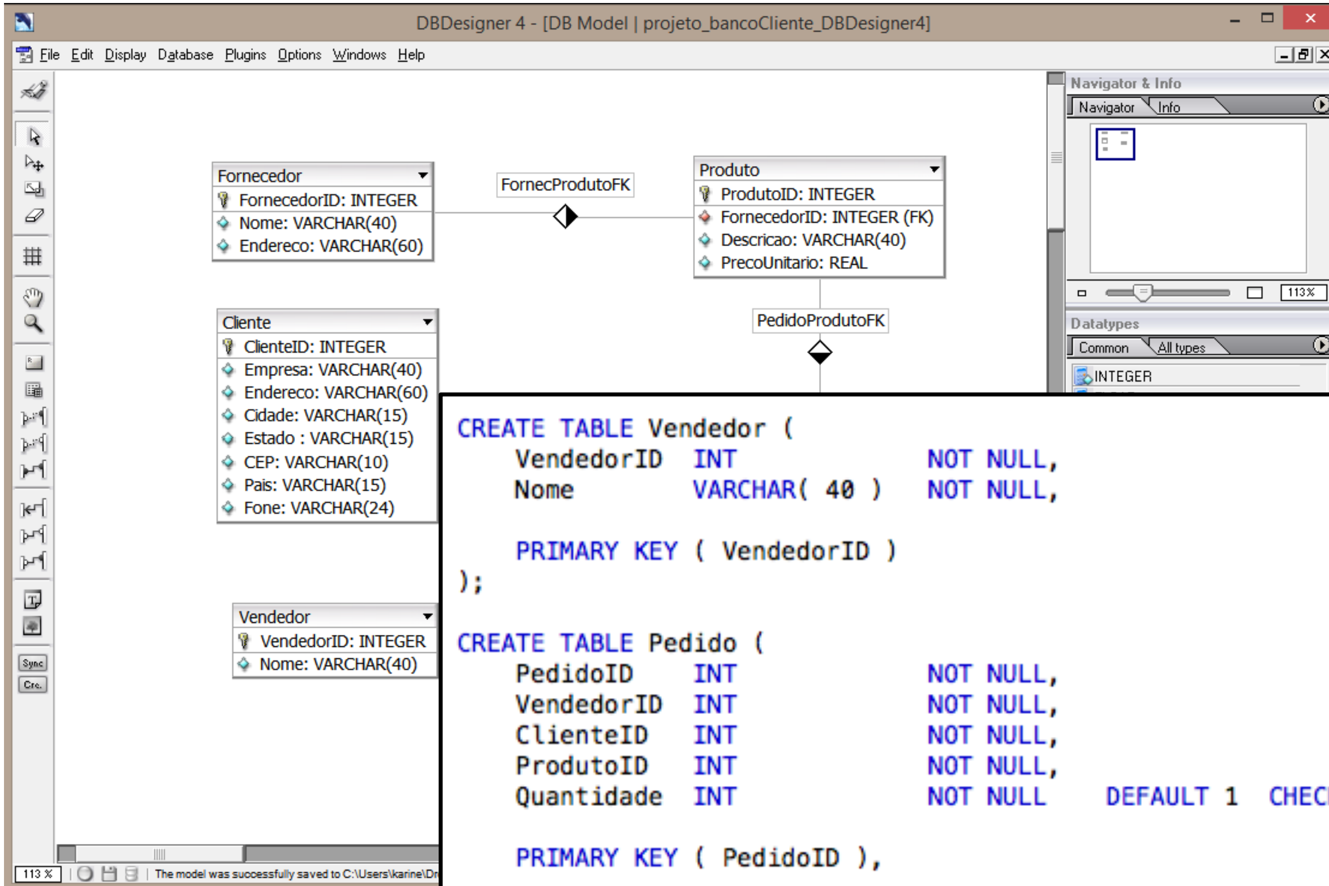
Diagrama criado com o aplicativo DBDesigner 4.

DBDesigner 4

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



THE MYSQL DATA-BASE DESIGNER
DBDesigner 4




```sql
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 relational diagrams

MySQL Workbench

- ✓ MySQL Workbench is a graphical tool for working with MySQL Servers and databases. It is the successor of DBDesigner 4.
- ✓ MySQL Workbench Commercial and MySQL Workbench Community (free)
- ✓ https://www.mysql.com/products/workbench/
- ✓ Developed and optimized for the open source MySQL-Database, but it can create standard SQL scripts from its diagrams

SQL: Structured Query Language

- A standard (ISO) for relational databases.
- Based on the *relational algebra*
- *Higher-level declarative language* interface: user only specifies what the result is to be, leaving the actual optimization and decisions on how to execute the query to the DBMS.
- Statements for:
  - ✓ data definitions, queries, and updates: DDL and DML
  - ✓ defining views on the database
  - ✓ specifying security and authorization
  - ✓ defining integrity constraints, and
  - ✓ specifying transaction controls.

Relational Algebra

- Defines a set of operations for the relational model.
- Its operations can be divided into two groups:
  1) **Set** operations, including UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT
  2) Operations for **relational databases**, including SELECT, PROJECT, and JOIN
- **Unary** operations (single relation) x **binary** operations (two relations)

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Source: (Elmasri and Navathe, 2011)

Unary Operation: SELECT

SELECT operation is used to choose a subset of the tuples from a relation that satisfies a selection condition. Symbol: sigma.

$$\sigma_{<\text{selection condition}>}(R)$$

$$\sigma_{(\text{Dno}=4 \ \textbf{AND} \ \text{Salary}>25000) \ \textbf{OR} \ (\text{Dno}=5 \ \textbf{AND} \ \text{Salary}>30000)}(\text{EMPLOYEE})$$

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

Unary Operation: PROJECT

PROJECT operation selects certain columns from the table and discards the other columns. Symbol: Pi.

$$\pi_{\text{<attribute list>}}(R)$$

$$\pi_{\text{Sex, Salary}}(\text{EMPLOYEE})$$

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

PROJECT and SELECT

$$\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno}=5}(\text{EMPLOYEE}))$$

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Set Operation

- UNION (R ∪ S): the result is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
- INTERSECTION (R ∩ S) : The result is a relation that includes all tuples that are in both R and S.
- SET DIFFERENCE or MINUS (R – S): The result is a relation that includes all tuples that are in R but not in S.

The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Source: (Elmasri and Navathe, 2011)

CARTESIAN PRODUCT - CROSS PRODUCT

CARTESIAN PRODUCT ($R \times S$) : produces a new relation by combining every member (tuple) from one relation R (set) with every member (tuple) from the other relation S (set).

EMP_DEPENDENTS ← EMPNAMES X DEPENDENT

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Source: (Elmasri and Navathe, 2011)

EMP_DEPENDENTS ← EMPNAMES X DEPENDENT

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	. . .
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	. . .
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	. . .
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	. . .
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	. . .
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	. . .
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	. . .
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	. . .
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	. . .
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	. . .
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	. . .
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	. . .
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	. . .
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	. . .
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	. . .
Joyce	English	453453453	333445555	Alice	F	1986-04-05	. . .
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	. . .
Joyce	English	453453453	333445555	Joy	F	1958-05-03	. . .
Joyce	English	453453453	987654321	Abner	M	1942-02-28	. . .
Joyce	English	453453453	123456789	Michael	M	1988-01-04	. . .
Joyce	English	453453453	123456789	Alice	F	1988-12-30	. . .
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	. . .

JOIN Operation

JOIN operation is used to combine related tuples from two relations into single "longer" tuples. This operation is very important for any relational database because it allows us to process relationships among relations.

$$R \bowtie_{\langle \text{join condition} \rangle} S$$

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn}=\text{Ssn}} \text{EMPLOYEE}.$$

Source: (Elmasri and Navathe, 2011)

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Source: (Elmasri and Navathe, 2011)

JOIN Operation

JOIN operation is used to combine related tuples from two relations into single "longer" tuples. This operation is very important for any relational database because it allows us to process relationships among relations.

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE}.$$

DEPT_MGR

Dname	Dnumber	Mgr_ssn	. . .	Fname	Minit	Lname	Ssn	. . .
Research	5	333445555	. . .	Franklin	T	Wong	333445555	. . .
Administration	4	987654321	. . .	Jennifer	S	Wallace	987654321	. . .
Headquarters	1	888665555	. . .	James	E	Borg	888665555	. . .

Source: (Elmasri and Navathe, 2011)

JOIN Operation

$$((\text{PROJECT} \bowtie_{\text{Dnum}=\text{Dnumber}} \text{DEPARTMENT}) \bowtie_{\text{Mgr\_ssn}=\text{Ssn}} \text{EMPLOYEE})$$

Source: (Elmasri and Navathe, 2011)

JOIN Operation

$$((\text{PROJECT} \bowtie_{\text{Dnum=Dnumber}} \text{DEPARTMENT}) \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$$

- ✓ EQUIJOIN: join condition with only equality comparisons.
- ✓ THETA JOIN: any join condition.

Source: (Elmasri and Navathe, 2011)

DIVISION Operation

DIVISION operation is applied to two relations $R(Z) \div S(X)$, where the attributes of R are a subset of the attributes of S; that is, $X \subseteq Z$.

The result is a relation T. For a tuple *t* of R to appear in the result T, the values in *t* must appear in R in combination with every tuple in S.

Source: (Elmasri and Navathe, 2011)

$$SSNS(Ssn) \leftarrow SSN\_PNOS \div SMITH\_PNOS$$

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

Source: (Elmasri and Navathe, 2011)

$$SSNS(Ssn) \leftarrow SSN\_PNOS \div SMITH\_PNOS$$

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

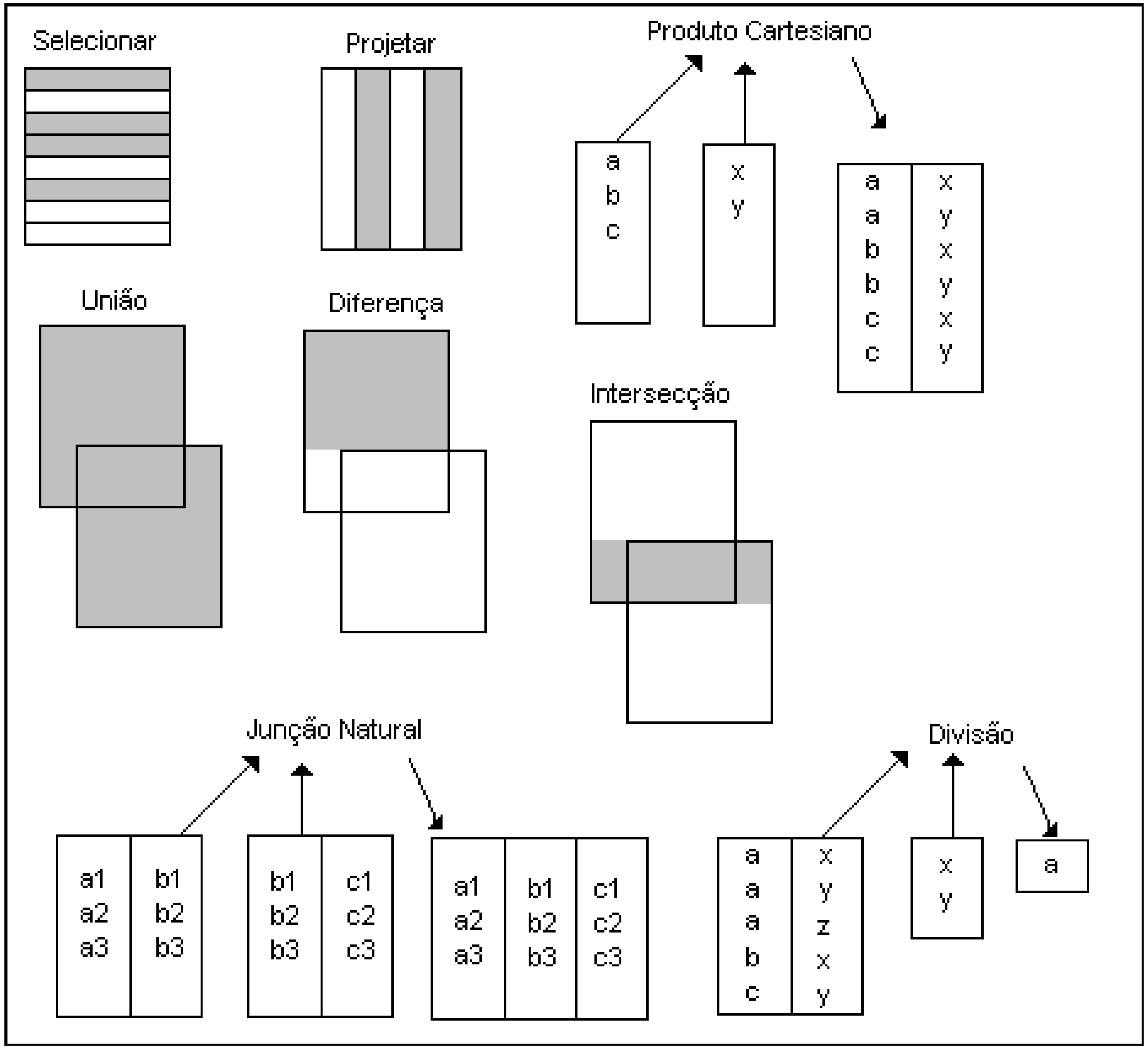
Pno
1
2

SSNS

Ssn
123456789
453453453

Source: (Elmasri and Navathe, 2011)

Relational Algebra - Summary



Relational Algebra - Summary

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$

Relational Algebra - Summary

UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

From Relational Algebra to SQL

SELECT operation => WHERE clause of a query.

$$\sigma_{(\text{Dno}=4\ \textbf{AND}\ \text{Salary}>25000)\ \textbf{OR}\ (\text{Dno}=5\ \textbf{AND}\ \text{Salary}>30000)}(\text{EMPLOYEE})$$

⇩

```
SELECT  *
FROM    EMPLOYEE
WHERE   (Dno = 4 AND Salary > 25000) OR
        (Dno = 5 AND Salary > 30000)
```

From Relational Algebra to SQL

PROJECT operation => SELECT clause of a query.

$$\pi_{\text{Sex, Salary}}(\text{EMPLOYEE})$$

```
SELECT DISTINCT Sex, Salary
FROM   EMPLOYEE
```

From Relational Algebra to SQL

CARTESIAN PRODUCT operation => FROM clause of a query.

EMPNAMES X DEPENDENT

```
SELECT *
FROM   EMPNAMES, DEPENDENT
```

From Relational Algebra to SQL

JOIN operation => FROM clause of a query.

$$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{Mgr\_ssn=Ssn} EMPLOYEE.$$

```
SELECT *
FROM   DEPARTMENT INNER JOIN EMPLOYEE
       ON Mgr_ssn = Ssn
```

From Relational Algebra to SQL

JOIN operation => FROM + WHERE clause of a query.

$$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{Mgr\_ssn=Ssn} EMPLOYEE.$$

⇩

```
SELECT *
FROM   DEPARTMENT, EMPLOYEE
WHERE  Mgr_ssn = Ssn
```

SQL: Structured Query Language

- SEQUEL: Originally, SQL was called SEQUEL (Structured English QUEry Language) - database system called SYSTEM R (IBM)
- SQL (ANSI 1986): called SQL-86 or SQL1, standard language for commercial relational DBMSs – ANSI and ISO.
- SQL-92 (also referred to as SQL2).
- SQL:1999, which started out as SQL3.
- SQL:2003 and SQL:2006: added XML features.
- SQL:2008: object database features in SQL

SQL: Structured Query Language

- SQL uses the terms **table**, **row**, and **column** for the formal relational model terms **relation**, **tuple**, and **attribute**
- Statements for:
  - ✓ data definitions, queries, and updates: DDL and DML
  - ✓ defining views on the database
  - ✓ specifying security and authorization
  - ✓ defining integrity constraints, and
  - ✓ specifying transaction controls.

SQL DDL – Data Definition Language

Examples of SQL DDL statements:

CREATE DATABASE – cria um novo banco de dados

ALTER DATABASE – modifica um banco de dados

CREATE SCHEMA – cria um novo esquema

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 DML – Data Manipulation Language

Examples of SQL DML statements:

SELECT – seleciona 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 Schema – Example

- An SQL schema groups together tables and other constructs that belong to the same database application.
- An SQL schema is identified by a schema **name**, and includes an **authorization identifier** to indicate the user or account who owns the schema.

```
CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith'
```

SQL – Create Table

```
CREATE TABLE [<schema name>.]<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`

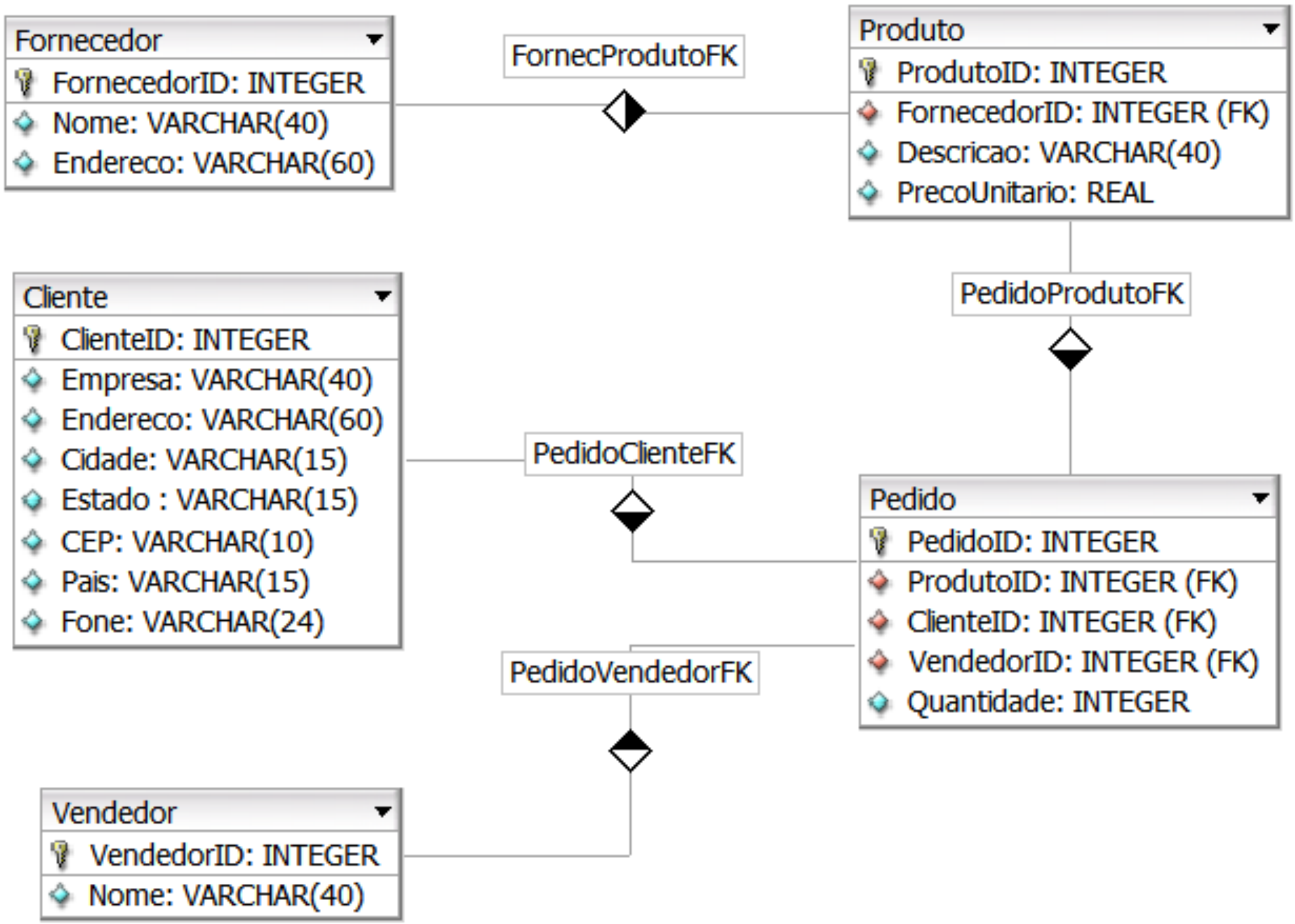
SQL – Constraints

<table>
<tr><td>Restringir que um atributo não tenha valores nulos</td><td>NOT NULL</td></tr>
<tr><td>Restringir valores e domínios de atributos</td><td>CHECK (<expression>)</td></tr>
<tr><td>Restringir que um ou mais atributos tenham valores únicos</td><td>UNIQUE (<column_name> {,<column_name>})</td></tr>
<tr><td>Definir chave primária</td><td>PRIMARY KEY (<column_name> {,<column_name>})</td></tr>
<tr><td>Definir restrições de integridade referencial (chave estrangeira)</td><td>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)</td></tr>
</table>

SQL – Referential Triggered Action

- ✓ SET NULL: if a tuple of a supervising table is deleted / updated, the value of all tuples that were referencing it are automatically set to NULL.
- ✓ SET DEFAULT: if a tuple of a supervising table is deleted / updated, the value of all tuples that were referencing it are automatically set to their default values.
- ✓ CASCADE: if a tuple of a supervising table is deleted / updated, the value of all tuples that were referencing it are automatically deleted or updated to the new value.

SQL – Create table – Examples



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 – 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* ou *junção* 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 – Aggregation Functions

- 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?

```sql
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.

```sql
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 – Select-From-Where – Examples

- ✓ Selecione todos os vendedores que tem o nome ‘José’ em seu nome?

```
SELECT *
FROM vendedor
WHERE nome LIKE '%José%'
```

SQL – Create View

Virtual table or *View* is derived from other tables and does not necessarily exist in physical form.

A view is supposed to be always up-to-date; if we modify the tuples in the base tables on which the view is defined, the view must automatically reflect these changes.

```
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

```sql
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

SQL – Assertion

- ✓ CREATE ASSERTION: used to specify additional types of constraints that are outside the scope of the built-in relational model constraints.
- ✓ The DBMS is responsible for ensuring that the condition is not violated.
- ✓ Example: *the salary of an employee must not be greater than the salary of the manager of the department that the employee works*

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Source: (Elmasri and Navathe, 2011)

SQL – Assertion

Example: *the salary of an employee must not be greater than the salary of the manager of the department that the employee works*

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS  ( SELECT    *
                      FROM      EMPLOYEE E, EMPLOYEE M,
                                DEPARTMENT D
                      WHERE     E.Salary>M.Salary
                                AND E.Dno=D.Dnumber
                                AND D.Mgr_ssn=M.Ssn ) );
```

The DBMS is responsible for ensuring that the condition is not violated. Whenever some tuples in the database cause the condition of an ASSERTION statement to evaluate to FALSE, the constraint is violated.

SQL – Trigger

- ✓ CREATE TRIGGER: used to specify automatic actions that the database system will perform when certain events and conditions occur.
- ✓ Triggers can be used in various applications, such as maintaining database consistency, monitoring database updates, and updating derived data automatically.
- ✓ Example: *check whenever an employee's salary is greater than the salary of his or her direct supervisor*.

SQL – Trigger

Example: *check whenever an employee's salary is greater than the salary of his or her direct supervisor*.

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
    ON EMPLOYEE
FOR EACH ROW
    WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
                          WHERE SSN = NEW.SUPERVISOR_SSN ) )
                          INFORM_SUPERVISOR(NEW.Supervisor_ssn,
                          NEW.Ssn );
```

SQL – Trigger

Example: *check whenever an employee's salary is greater than the salary of his or her direct supervisor*.

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
    ON EMPLOYEE
FOR EACH ROW
    WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
                          WHERE SSN = NEW.SUPERVISOR_SSN ) )
                          INFORM_SUPERVISOR(NEW.Supervisor_ssn,
                          NEW.Ssn );
```

Events: before inserting a new employee record, changing an employee's salary, or changing an employee's supervisor.

Keyword: BEFORE or AFTER.

SQL – Trigger

Example: *check whenever an employee's salary is greater than the salary of his or her direct supervisor*.

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
    ON EMPLOYEE
FOR EACH ROW
    WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
                          WHERE SSN = NEW.SUPERVISOR_SSN ) )
                          INFORM_SUPERVISOR(NEW.Supervisor_ssn,
                          NEW.Ssn );
```

Condition: determines whether the rule action should be executed. The condition is specified in the WHEN clause of the trigger.

SQL – Trigger

Example: *check whenever an employee's salary is greater than the salary of his or her direct supervisor*.

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
    ON EMPLOYEE
FOR EACH ROW
    WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
                          WHERE SSN = NEW.SUPERVISOR_SSN ) )
                          INFORM_SUPERVISOR(NEW.Supervisor_ssn,
                          NEW.Ssn );
```

Action: a sequence of SQL statements or a database transaction or an external program.

SQL Procedural Language (PL/SQL)