Data Layer Design

Database Design

https://youtu.be/QkrELRavi80

Introduction

What is, and how do we get, persistence?

- **Persistence:**
  - ability that most software systems require for storing and obtaining data using a permanent storage system

- **Objects can be made persistent by using**:
  - Object-oriented data bases
  - Relational data bases
  - Object-relational data bases
  - XML files
  - etc.

Introduction

How does Data Base technology influence design?

Technological dependence:

- Properties that have to be satisfied (non-functional requirements)
- Technological resources available
  - Programming language paradigm
  - ***Data Base Management System (DBMS) paradigm***

Determine:

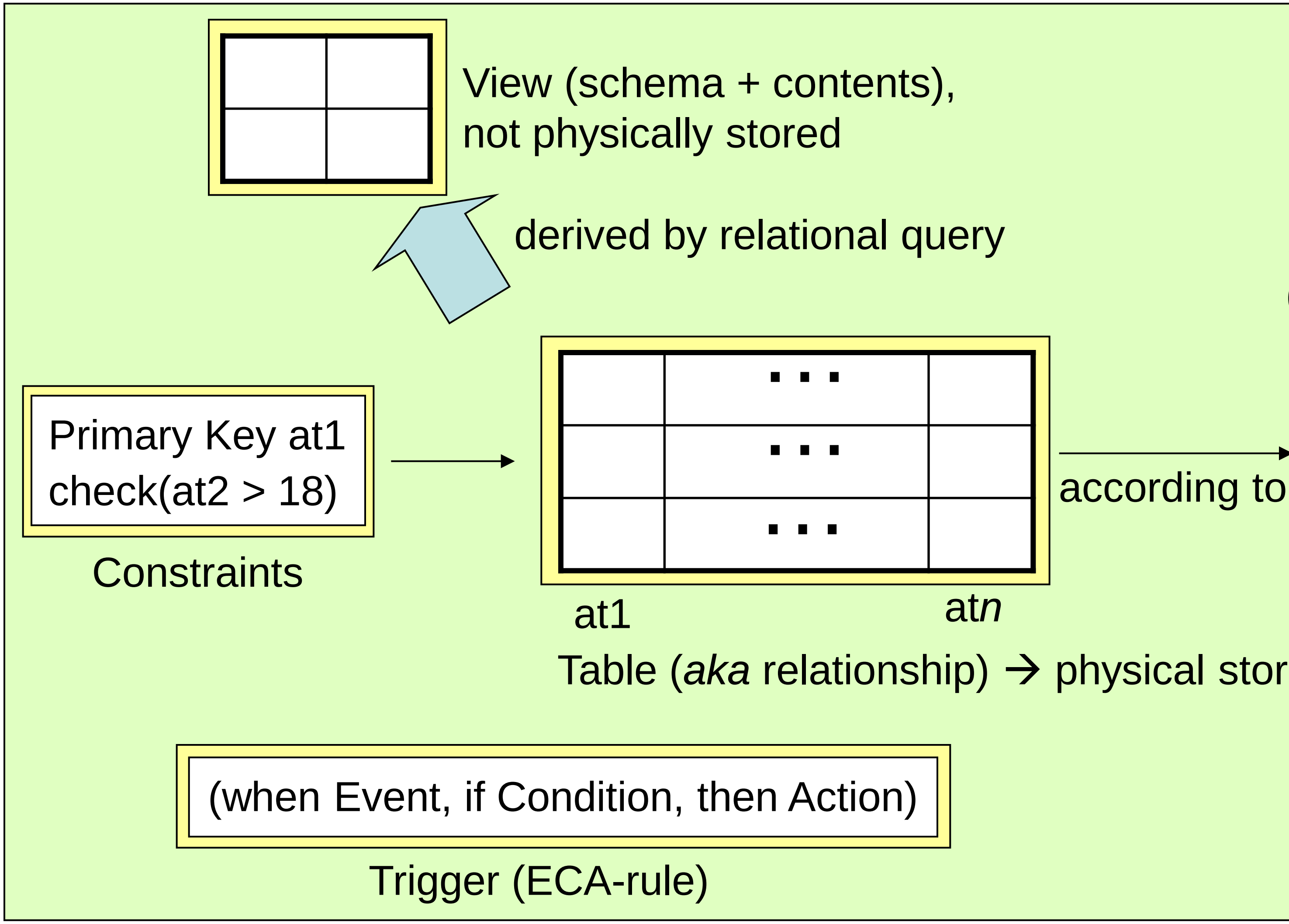
- The architecture of the software system and the patterns to be applied

Software design depends on the DBMS type used

Along this course we focus on the study
of persistence using relational DBMS

Relational Data Base Technology:

Data and Process components



View (schema + contents),
not physically stored

derived by relational query

Primary Key at1
check(at2 > 18)

Constraints

at1

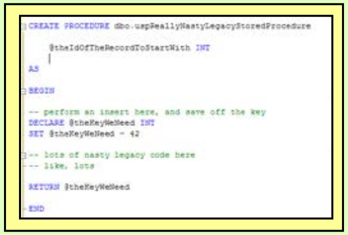
at*n*

Table (*aka* relationship) → physical storage

according to

(when Event, if Condition, then Action)

Trigger (ECA-rule)



```
CREATE PROCEDURE dbo.uspReallyNastyLegacyStoredProcedure

    @theIdOfTheRecordToStartWith INT

AS

BEGIN

-- perform an insert here, and save off the key
DECLARE @theKeyWeNeed INT
SET @theKeyWeNeed = 42

-- lots of nasty legacy code here
-- like, lots

RETURN @theKeyWeNeed

END
```

Stored procedure
(first class object)

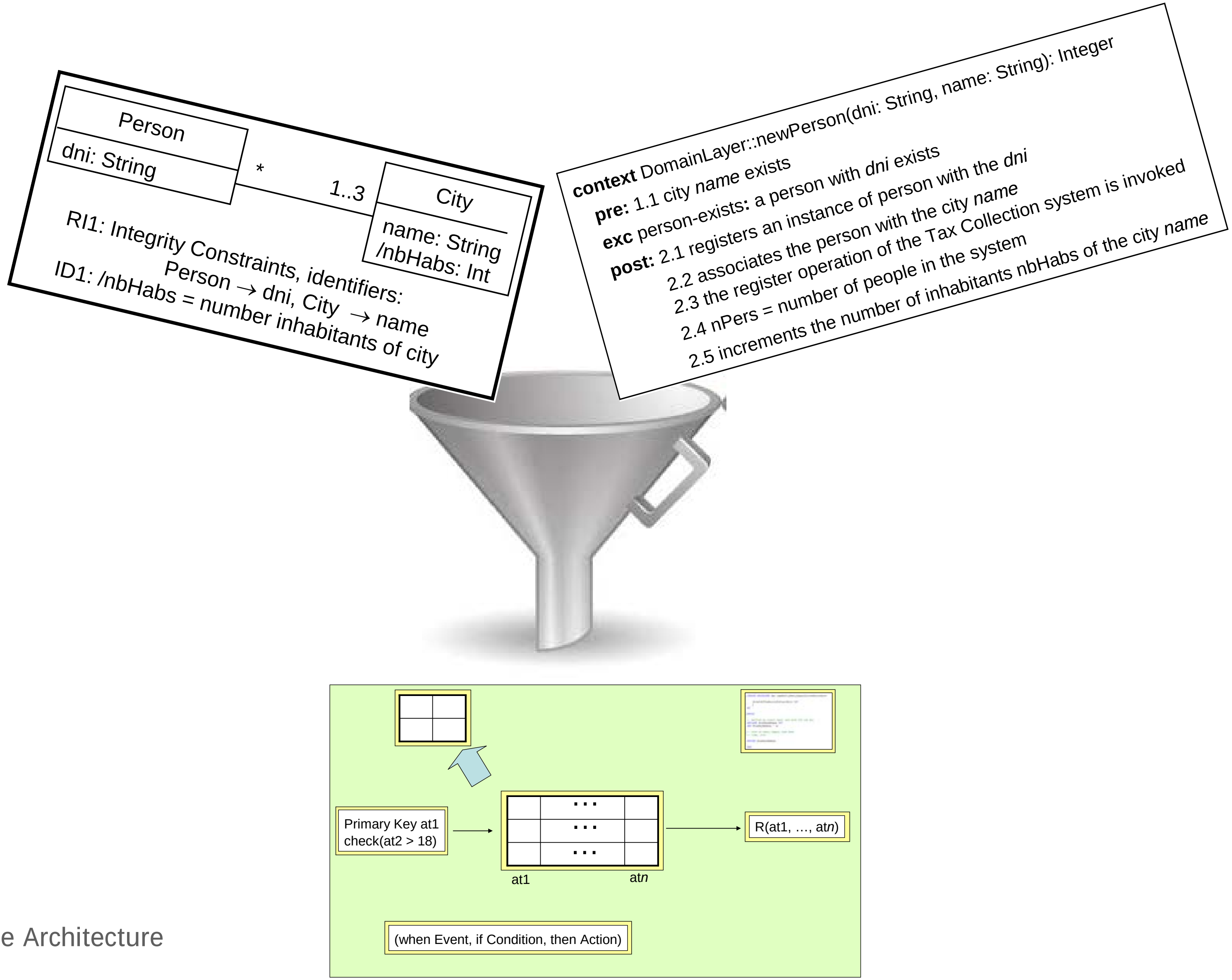
R(at1, …, at*n*)

Schema

Data base schema → grouping administrative unit

Relational Data Base Technology:

Logical Design of the DB Schema

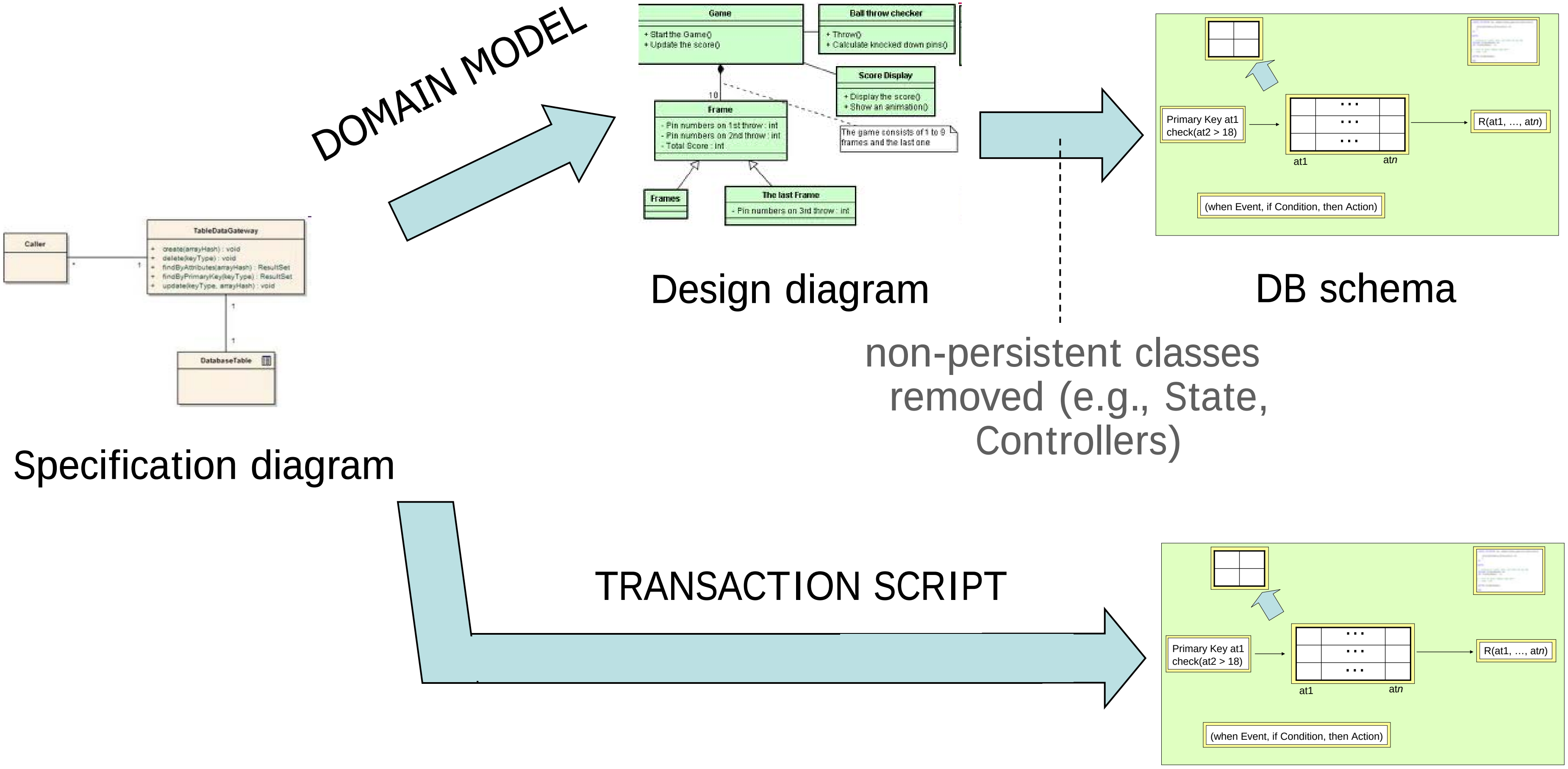


RI1: Integrity Constraints, identifiers:
Person → dni, City → name
ID1: /nbHabs = number inhabitants of city

context DomainLayer::newPerson(dni: String, name: String): Integer
pre: 1.1 city *name* exists
exc person-exists**:** a person with *dni* exists
post: 2.1 registers an instance of person with the *dni*
2.2 associates the person with the city *name*
2.3 the register operation of the Tax Collection system is invoked
2.4 nPers = number of people in the system
2.5 increments the number of inhabitants nbHabs of the city *name*

Relational Data Base Technology:

Logical Design of the DB Schema – Translation of the Class Diagram


DOMAIN MODEL
Game
+ Start the Game()
+ Update the score()
Ball throw checker
+ Throw()
+ Calculate knocked down pins()
Score Display
+ Display the score()
+ Show an animation()
10
Frame
- Pin numbers on 1st throw : int
- Pin numbers on 2nd throw : int
- Total Score : int
The game consists of 1 to 9 frames and the last one
Frames
The last Frame
- Pin numbers on 3rd throw : int
Caller
*
1
TableDataGateway
+ create(arrayHash) : void
+ delete(keyType) : void
+ findByAttributes(arrayHash) : ResultSet
+ findByPrimaryKey(keyType) : ResultSet
+ update(keyType, arrayHash) : void
1
1
DatabaseTable
Primary Key at1
check(at2 > 18)
. . .
. . .
. . .
at1
atn
R(at1, …, atn)
(when Event, if Condition, then Action)
TRANSACTION SCRIPT
Primary Key at1
check(at2 > 18)
. . .
. . .
. . .
at1
atn
R(at1, …, atn)
(when Event, if Condition, then Action)

Design diagram

DB schema

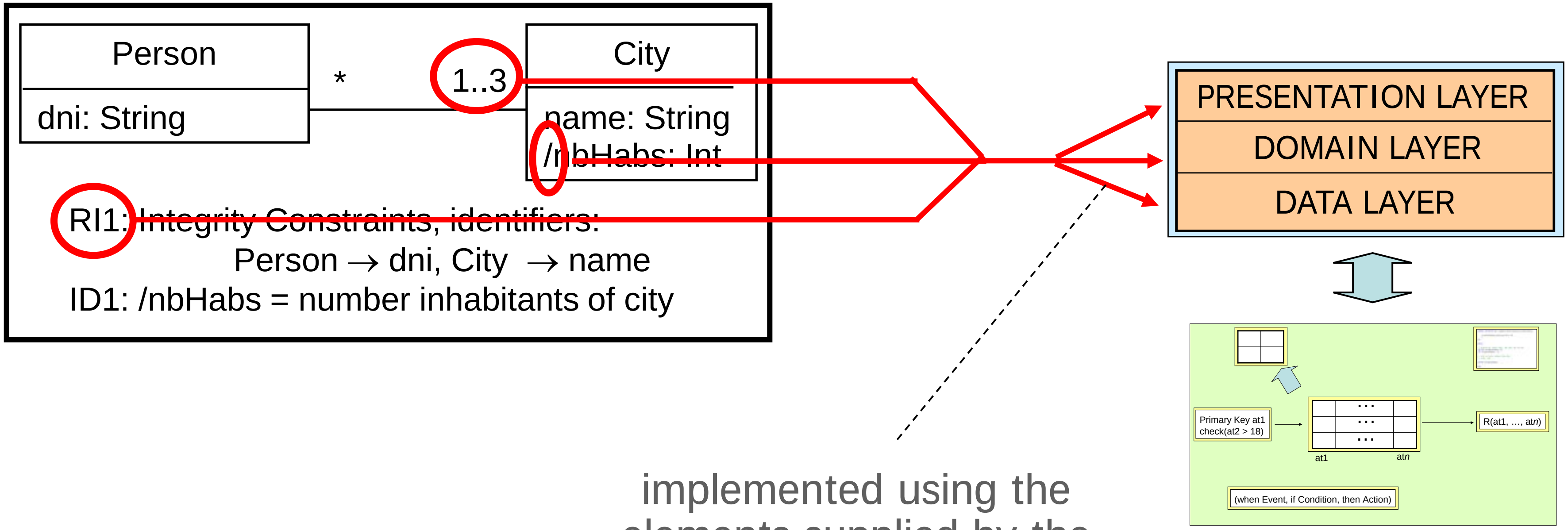
non-persistent classes removed (e.g., State, Controllers)

Specification diagram

DB schema

Relational Data Base Technology:

Logical Design of the DB Schema – Translation of other Responsibilities



Person

dni: String

*

1..3

City

name: String

/nbHabs: Int

RI1: Integrity Constraints, identifiers:

Person → dni, City → name

ID1: /nbHabs = number inhabitants of city

PRESENTATION LAYER

DOMAIN LAYER

DATA LAYER

implemented using the elements supplied by the relational technology (as done with identifier constraints)

Relational Data Base Technology:

Logical Design of the DB Schema – Translation of other Responsibilities

context DomainLayer::newPerson(dni: String, name: String): Integer

pre: 1.1 city *name* exists

exc person-exists**:** a person with *dni* exists

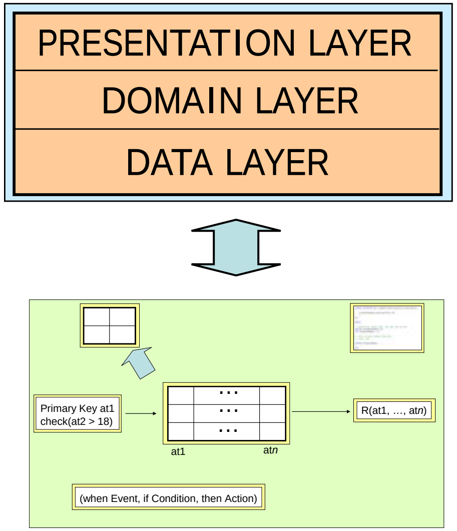
post: 2.1 registers an instance of person with the *dni*

2.2 associates the person with the city *name*

2.3 the register operation of the Tax Collection system is invoked

2.4 nPers = number of people in the system

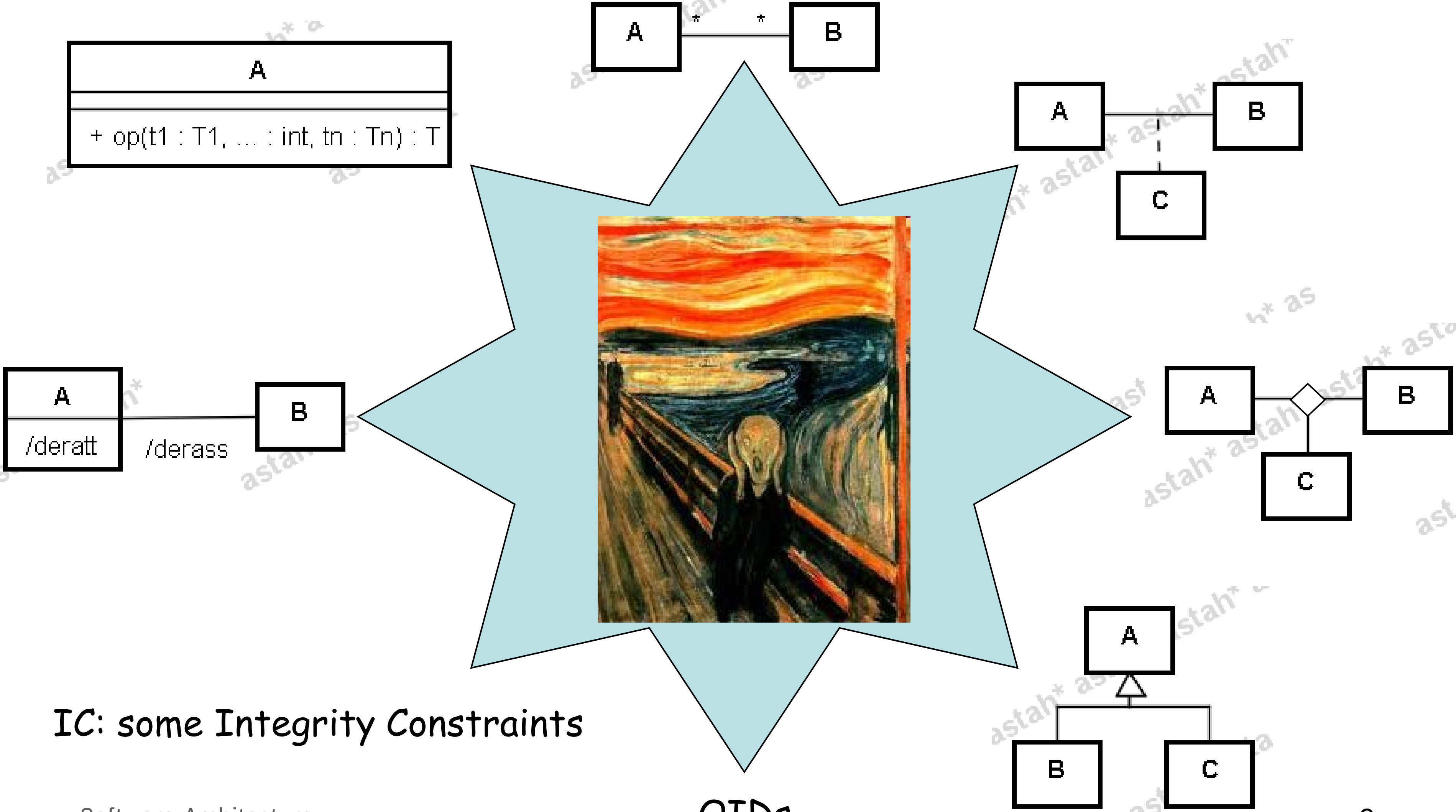
2.5 increments the number of inhabitants nbHabs of the city *name*



implemented using the elements supplied by the relational technology (as done with identifier constraints)

Relational Data Base Technology:

Logical Design of the DB Schema – Not Covered in Relational Technology

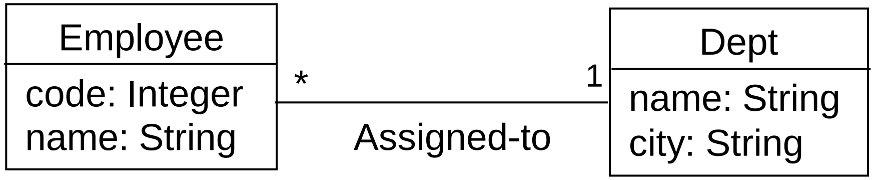


IC: some Integrity Constraints

OIDs

Relational Data Base Technology:

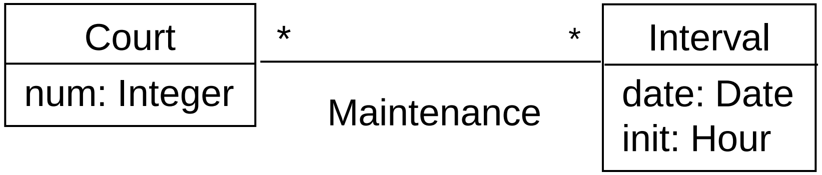
Logical Design of the DB, binary associations



Is translated into:

Dept (name-d, city)

Employee (code, name-emp, name-d)



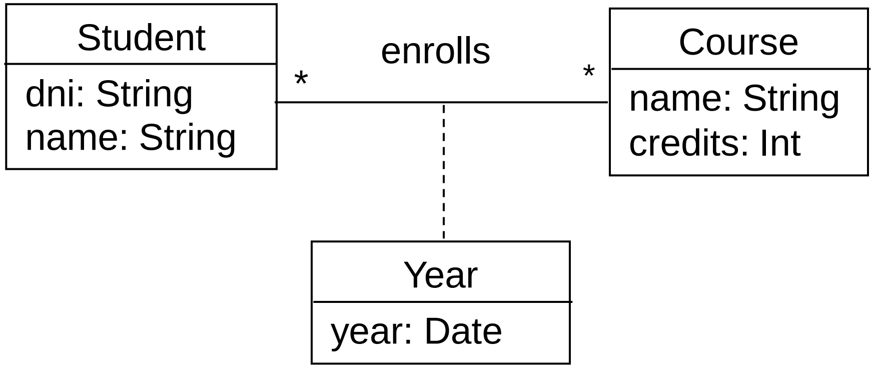
Is translated into:

Interval (date, init)

Court (num)

Maintenance (num-court, date, init)

Relational Data Base Technology:
Logical Design of the DB, class associations



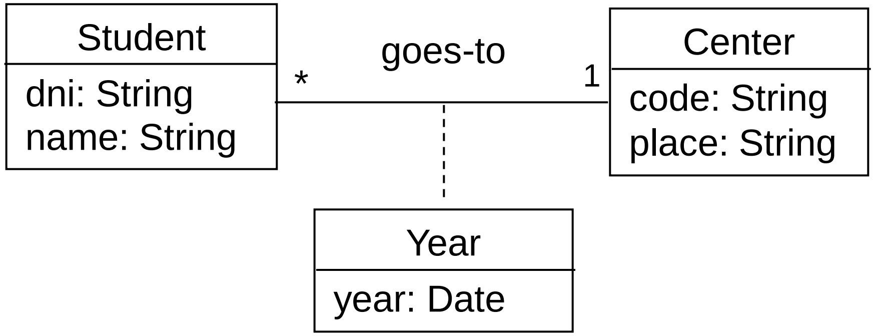
Is translated into:

Student (dni, name)

Course (name, credits)

Year(dni, name, year)

In the unlikely event that some class association has a role with multiplicity 1 (or 0..1), the table is created anyway:



Is translated into:

Student (dni, name, codeCenter)

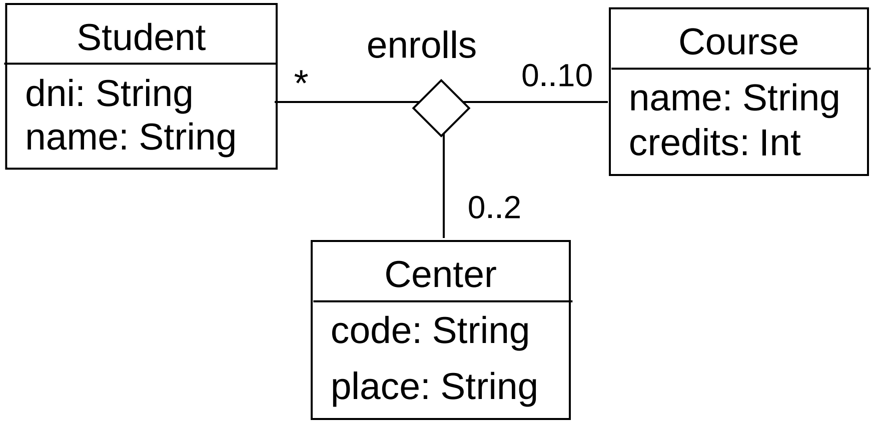
Center (code, place)

Year(dni, code, year)

Relational Data Base Technology:

Logical Design of the DB, n-ary associations, n > 2

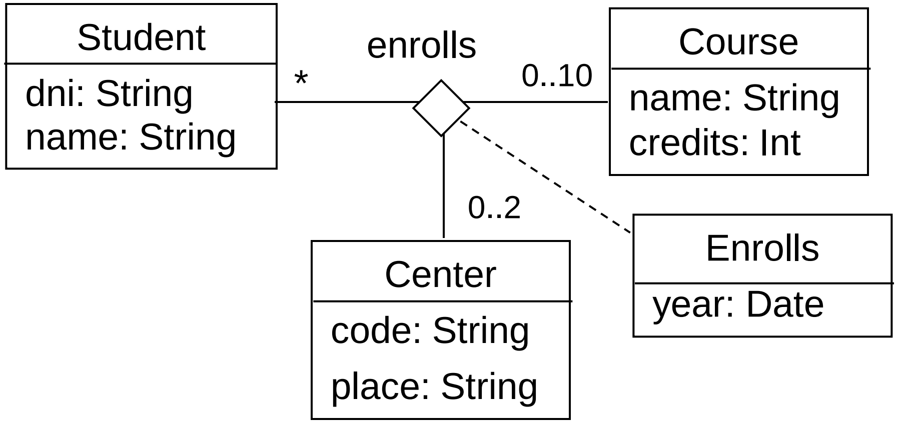
The treatment does not depend on multiplicities:



Is translated into:

Student (dni, name)
Course (name, credits)
Center(code, place)
enrolls(dni, name, code)

If it is class association, the treatment is the same:



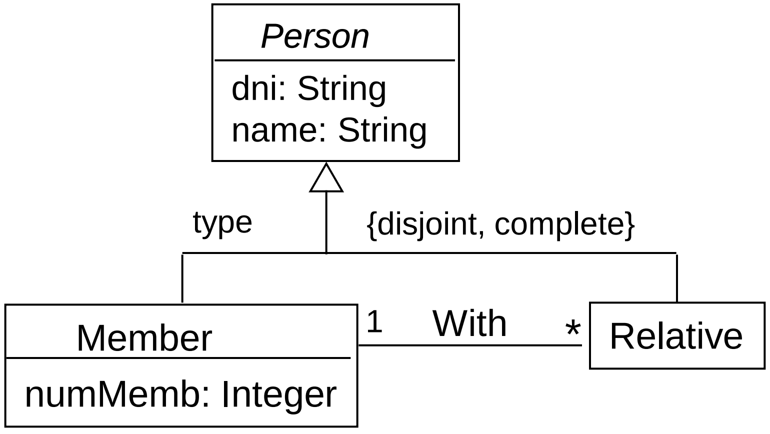
Is translated into:

Student (dni, name)
Course (name, credits)
Center(code, place)
enrolls(dni, name, code, year)

Relational Data Base Technology:

Logical Design of the DB, specialization hierarchies (1)

The translation depends on whether the hierarchy is collapsed or not



Textual Integrity Constraints

- Person Identifier: (Person, dni)
- Member Identifier: (Member, numMemb)

may appear or not

***Class Table Inheritance*:**

Person (dni, name, type)

Member (dni, numMemb)

Relative (dni, member)

alternative key

***Concrete Table Inheritance*:**

Member (dni, numMemb, name)

Relative (dni, name, member)

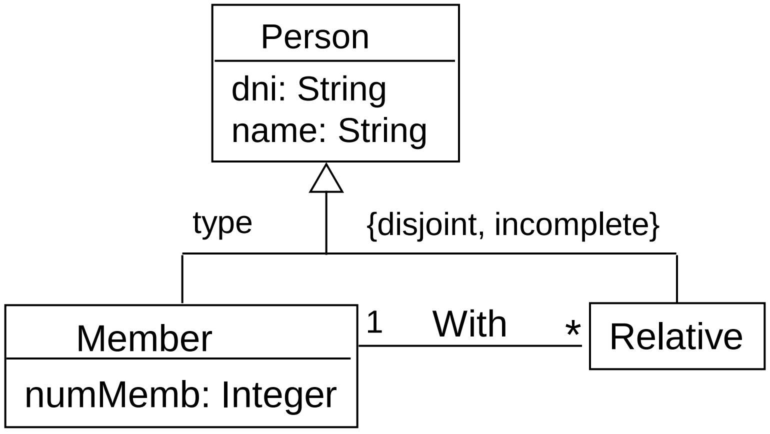
***Single Table Inheritance*:**

Person (dni, name, type, numMemb, member)

Accept null values

Relational Data Base Technology:
Logical Design of the DB, specialization hierarchies (1, variant)

The translation depends on whether the hierarchy is collapsed or not



Textual Integrity Constraints

- Person Identifier: (Person, dni)
- Member Identifier: (Member, numMemb)

may appear or not

***Class Table Inheritance*:**

Person (dni, name, type)

Member (dni, numMemb)

Relative (dni, member)

alternative key

***Concrete Table Inheritance*:**

Person (dni, name)

Member (dni, numMemb, name)

Relative (dni, name, member)

***Single Table Inheritance*:**

Person (dni, name, type, numMemb, member)

Accept null values

Relational Data Base Technology:
Logical Design of the DB, specialization hierarchies (2)

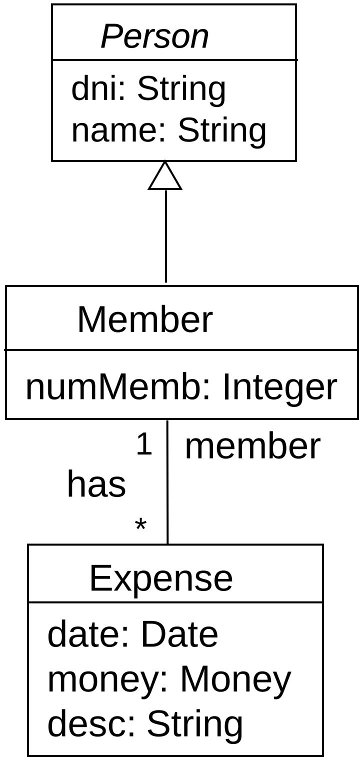
Strategy	Benefits	Drawbacks
Class Table Inheritance	Simple Changeable	Not much efficient (multiple accesses per object)
Concrete Table Inheritance	Efficient (one access per object)	Not much changeable (propagation of changes into abstract classes)
Single Table Inheritance	Efficient (one access per object) Changeable	Loss of space (but the DB may help)

There is not "the best" strategy. Talk to your Data Base Administrator!

Relational Data Base Technology:

Logical Design of the DB, classes without external keys

- It is necessary to add an “artificial” external key
  - Usually it will be a key maintained by the system itself



Textual Integrity Constraints

- Keys: (Person, dni), (Member, numMemb)

The table Expense looks like:

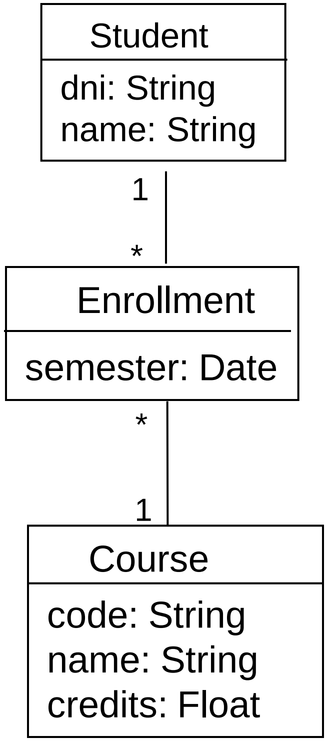
Expense(id-expense, date, payment, desc, member)

“artificial” identifier

Relational Data Base Technology:

Logical Design of the DB, classes with inconvenient keys

- Some keys are inconvenient to manage
  - The concept of key may change
  - The key is composed of several attributes



Textual Integrity Constraints

–Keys: (Student, dni), (Course, code)

–There cannot be more than one enrollment for a student and course a given semester

Non-functional requirements

–The system shall be prepared to future offering of courses to foreign students

Course(code, name, credits)

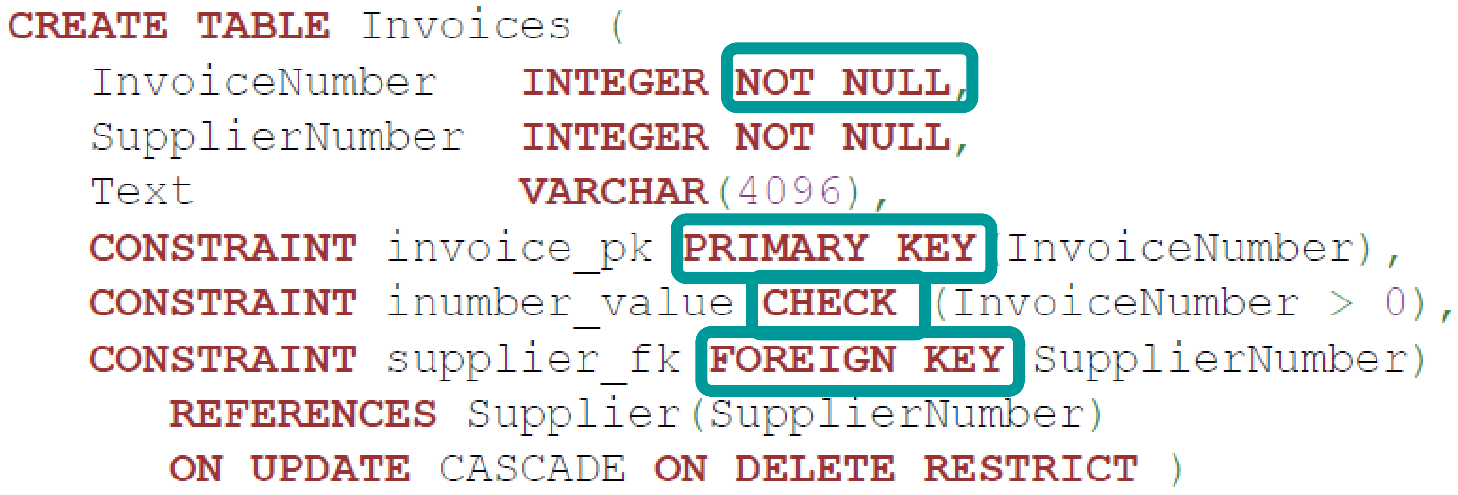
Student(id-student, dni, name)

Enrollment(id-enroll, id-student, code, semester)

Relational Data Base Technology:

Management of integrity constraints

- Relational DBMS provide diverse functionalities for dealing with integrity constraints:

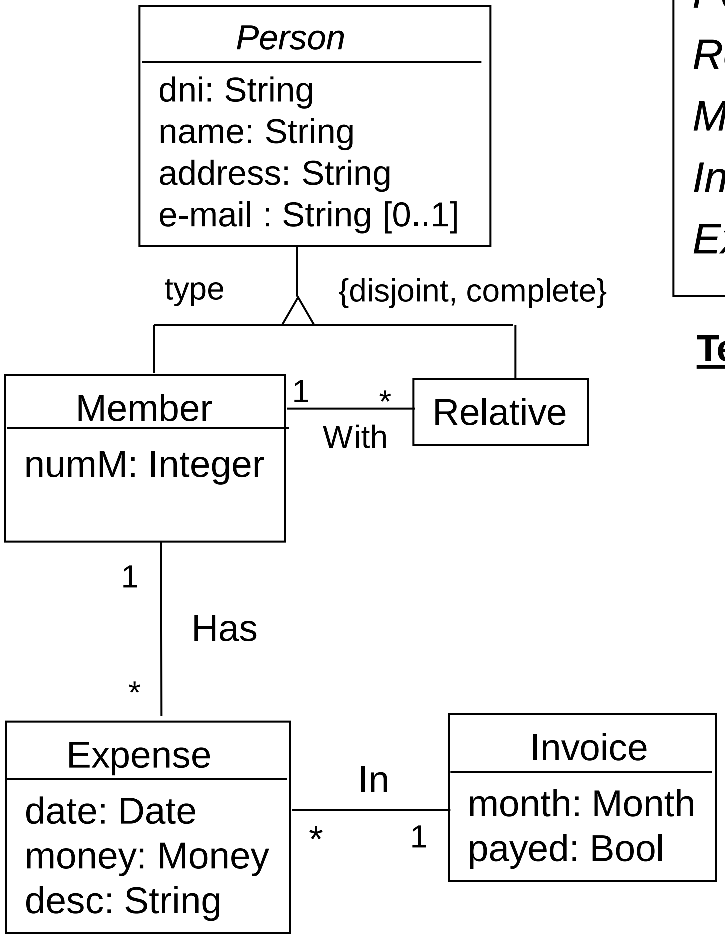


```
CREATE TABLE Invoices (
   InvoiceNumber    INTEGER NOT NULL,
   SupplierNumber   INTEGER NOT NULL,
   Text             VARCHAR(4096),
   CONSTRAINT invoice_pk PRIMARY KEY (InvoiceNumber),
   CONSTRAINT inumber_value CHECK (InvoiceNumber > 0),
   CONSTRAINT supplier_fk FOREIGN KEY (SupplierNumber)
      REFERENCES Supplier(SupplierNumber)
      ON UPDATE CASCADE ON DELETE RESTRICT )
```

The corresponding responsibilities may be directly assigned to the relational DBMS

Relational Data Base Technology:

Management of integrity constraints, example



Person (dni, name, address, e-mail)

Relative (dni, member)

Member (dni, numM*)*

Invoice (dni, month, payed)

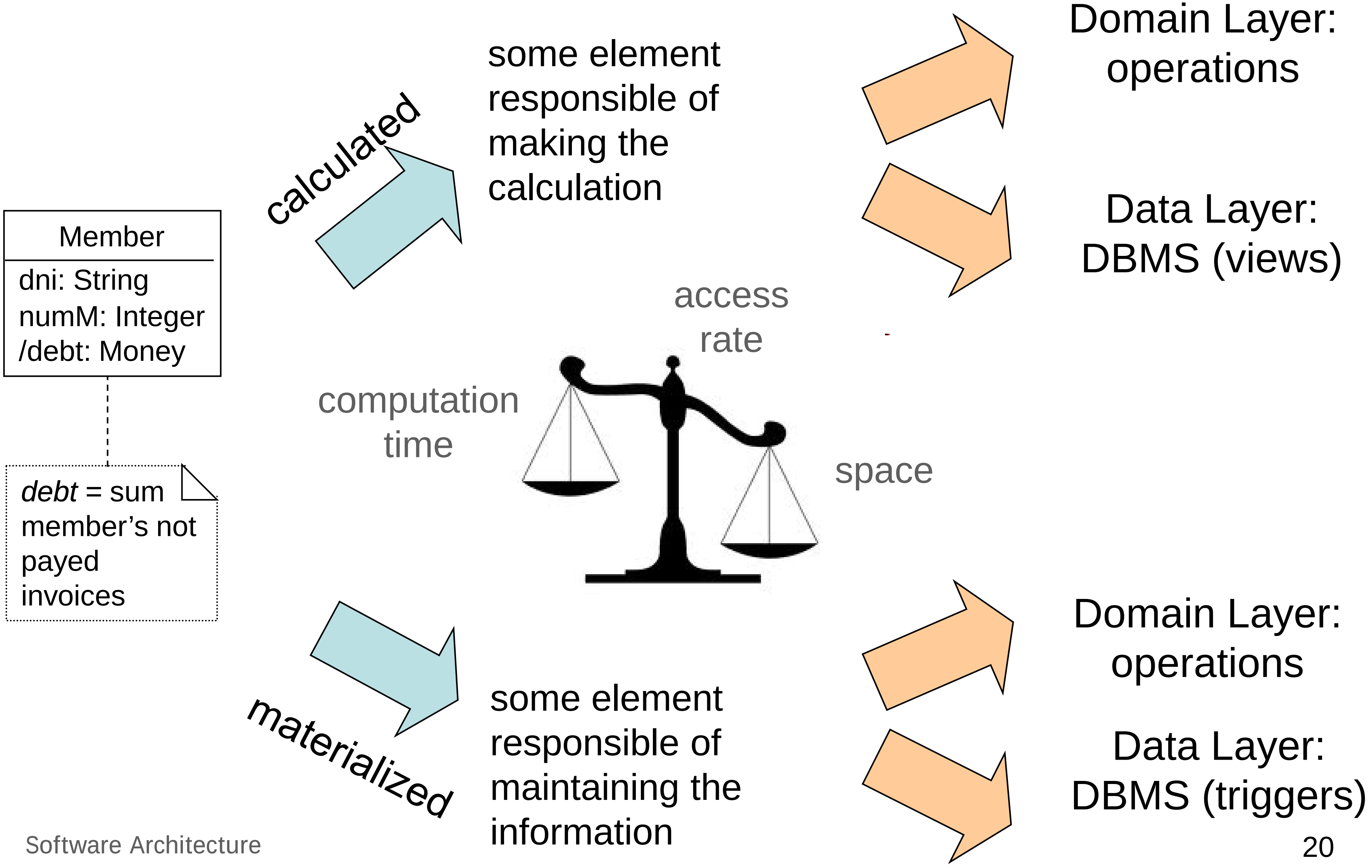
Expense (id-expense, date, money, desc, member, month)

Textual Integrity Constraints

- Identifiers: (Person, dni)
  - => Primary key of *Person*: *dni*
- Identifiers: (Member, numM)
  - => Primary key of *Member*: *dni* and alternative key, *numM*: unique(numM)
- There cannot be two invoices for the same member and month
  - => Primary key of *Invoice*: (*dni*, *month*)
- All the expenses of an invoice belong to the same month
  - => Check of *Expense*: check(month(data) = month)

Relational Data Base Technology:

Dealing with derived information



Member

dni: String
numM: Integer
/debt: Money

debt = sum member’s not payed invoices

calculated

some element responsible of making the calculation

Domain Layer: operations

Data Layer: DBMS (views)

access rate

computation time

space

materialized

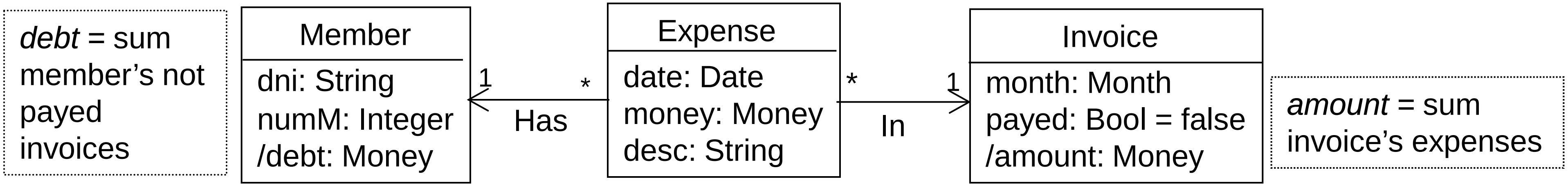
some element responsible of maintaining the information

Domain Layer: operations

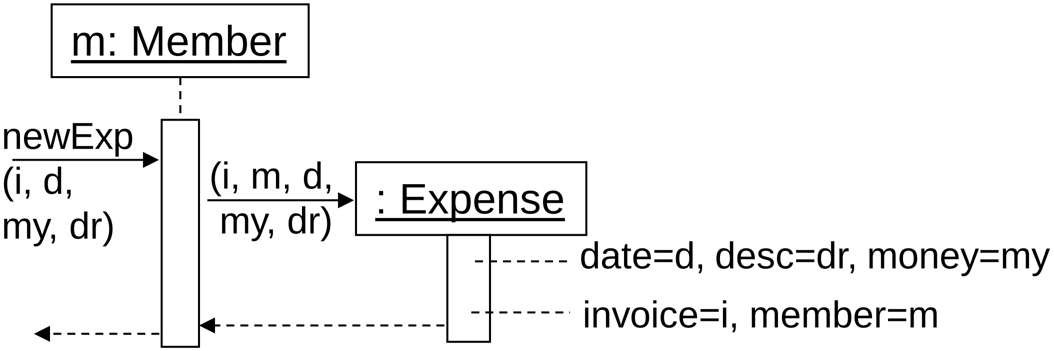
Data Layer: DBMS (triggers)

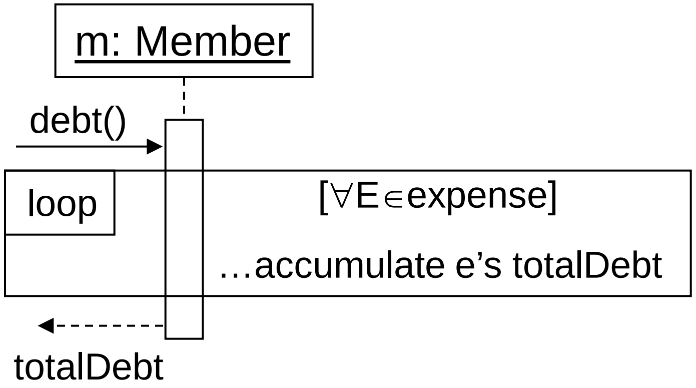
Relational Data Base Technology:

Dealing with derived information, calculated case
domain layer’s assignment



DOMAIN LAYER





DATA LAYER

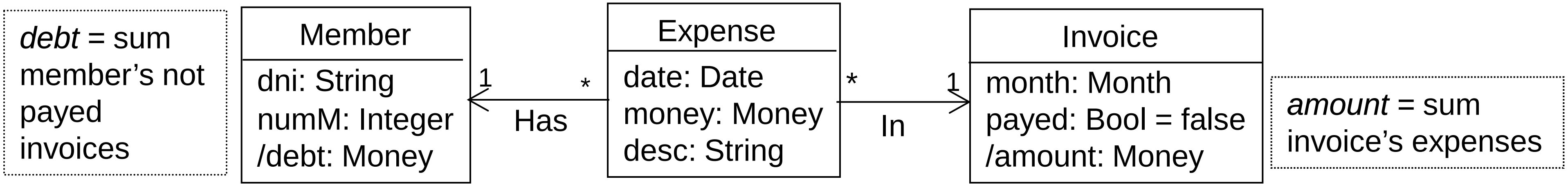
Member (dni, numM)

Invoice (dni, month, payed)

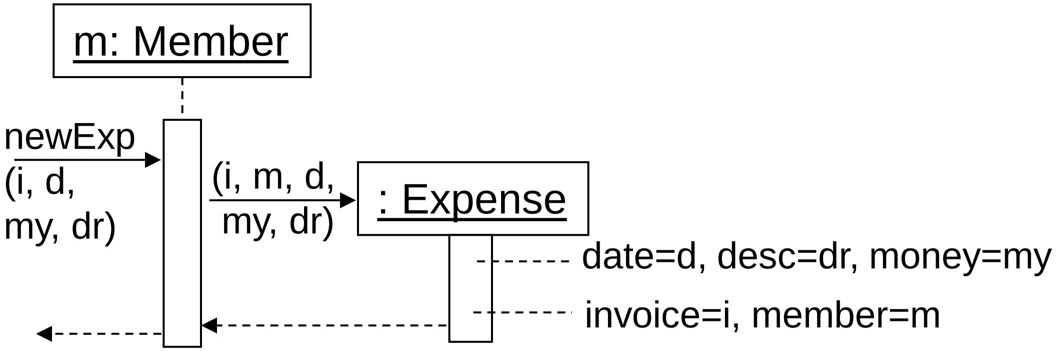
Expense (id-exp, date, money, desc,
member, month)

Relational Data Base Technology:

Dealing with derived information, calculated case
data layer’s assignment with views



DOMAIN LAYER



DATA LAYER

Member (dni, numM)

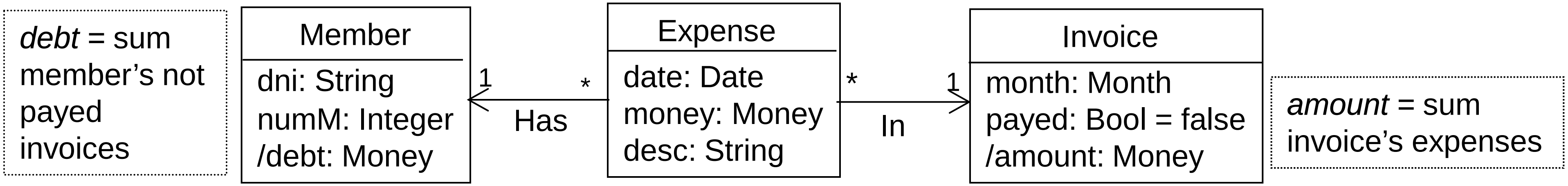
Invoice (dni, month, payed)

Expense (id-exp, date, money, desc,
member, month)

```
CREATE VIEW   debt [dni, debt]   AS
SELECT    s.dni, sum(c.money)
FROM      Member s, Expense c, Invoice r
WHERE     s.dni = c.member AND
          c.member = r.dni AND
          c.month = r.month AND
          r.payed='F'
GROUP BY dni
```

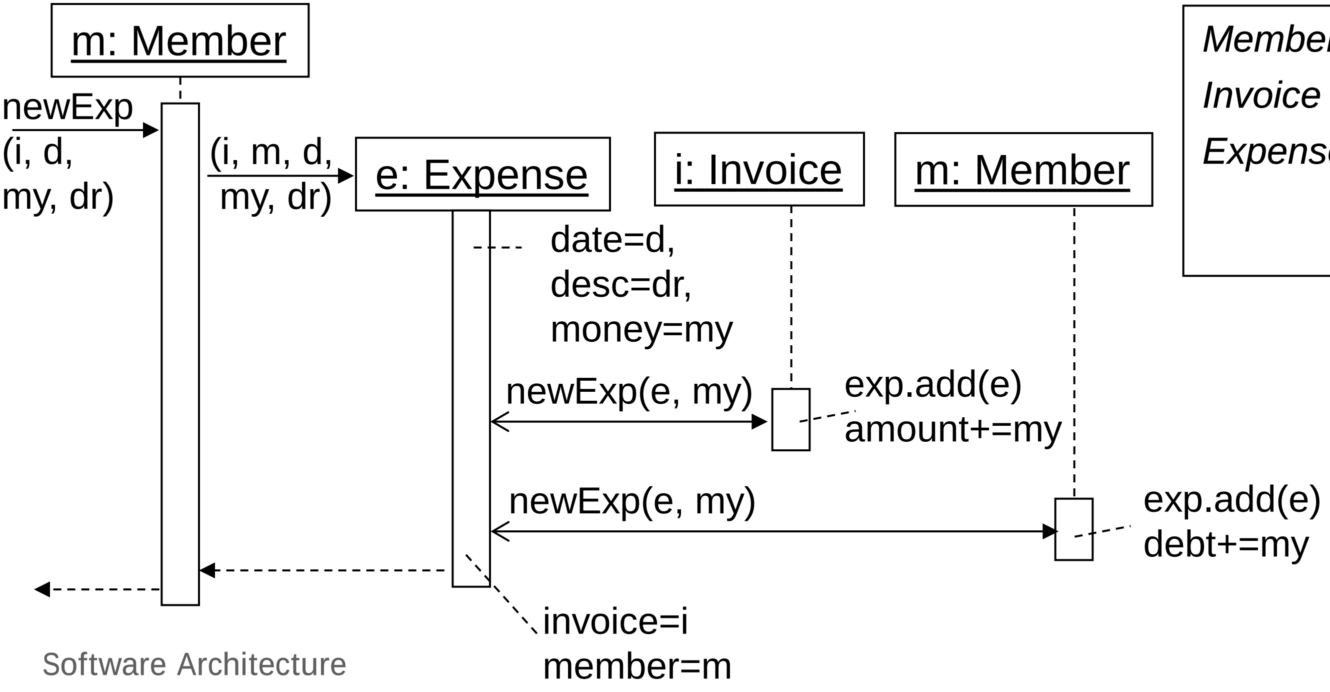
Relational Data Base Technology:

Dealing with derived information, materialized case domain layer's assignment



DOMAIN LAYER

DATA LAYER



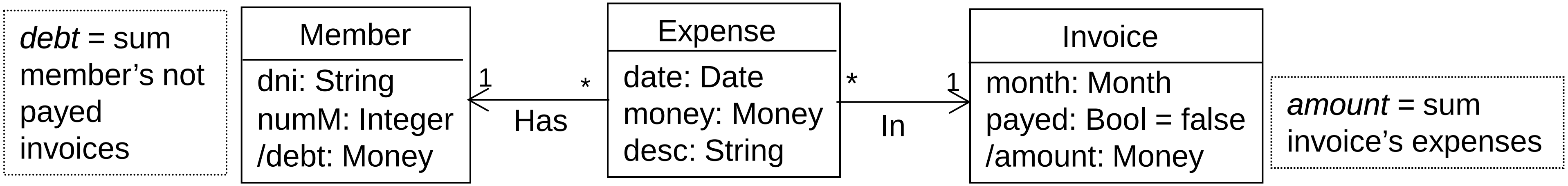
Member (dni, numM, debt)

Invoice (dni, month, payed, amount)

Expense (id-exp, date, money, desc, member, month)

Relational Data Base Technology:

Dealing with derived information, materialized case data layer’s assignment with triggers



debt = sum member’s not payed invoices

Member
dni: String
numM: Integer
/debt: Money

1 Has *

Expense
date: Date
money: Money
desc: String

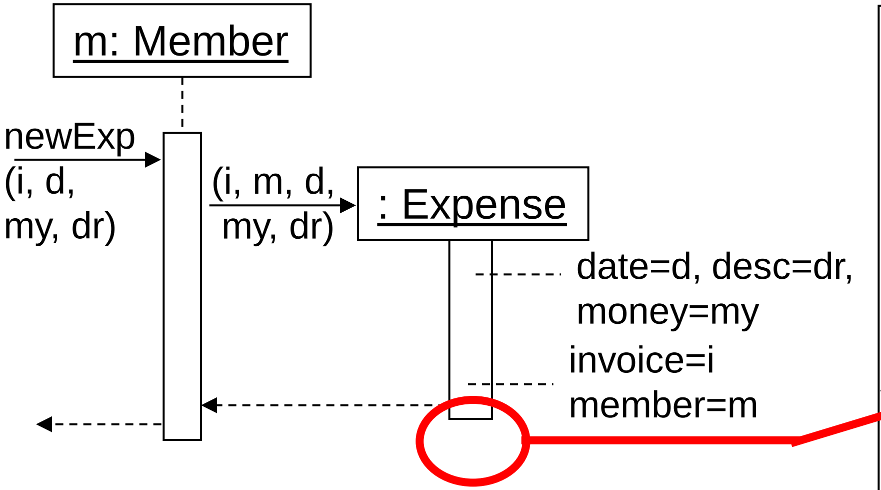
* In 1

Invoice
month: Month
payed: Bool = false
/amount: Money

amount = sum invoice’s expenses

DOMAIN LAYER

DATA LAYER



m: Member

newExp (i, d, my, dr)

(i, m, d, my, dr)

: Expense

date=d, desc=dr, money=my

invoice=i

member=m

Member (dni, numM, debt)

Invoice (dni, month, payed, amount)

Expense (id-exp, date, money, desc, member, month)

CREATE TRIGGER inc-money INSERT ON Expense
REFERENCING new AS nou
FOR EACH ROW (UPDATE Invoice r SET r.money =
r.money + nou.money
WHERE nou.dni = r.dni AND nou.month = r.month)

...*similar triggers for the rest…*

Relational Data Base Technology:

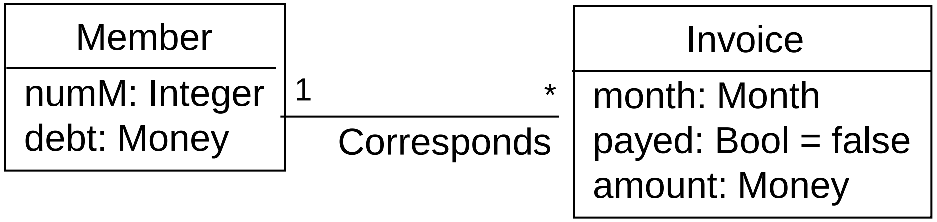
Operation design, assigning responsibilites to the DBMS

- Current specification languages do not allow defining active behaviour in conceptual schemas.
- On the contrary, relational DBMS do.
- Therefore, sometimes specification contracts define aspects that the DBMS may manage directly.

These responsibilities may be assigned to the DBMS

Relational Data Base Technology:

Operation design, assigning responsibilites to the DBMS



Member (dni, numM, debt)

Invoice (member, month, payed, amount)

context DomainLayer::removeMember(dniM: Integer)
exc member-not-exists: there is no member with dniM
post 2.1: the Member with dniM is removed
2.2: ***All the invoices of that Member are removed***

Assigned
to the
DBMS

```
CREATE TABLE Member (
    dni STRING PRIMARY KEY,
    ...)
```

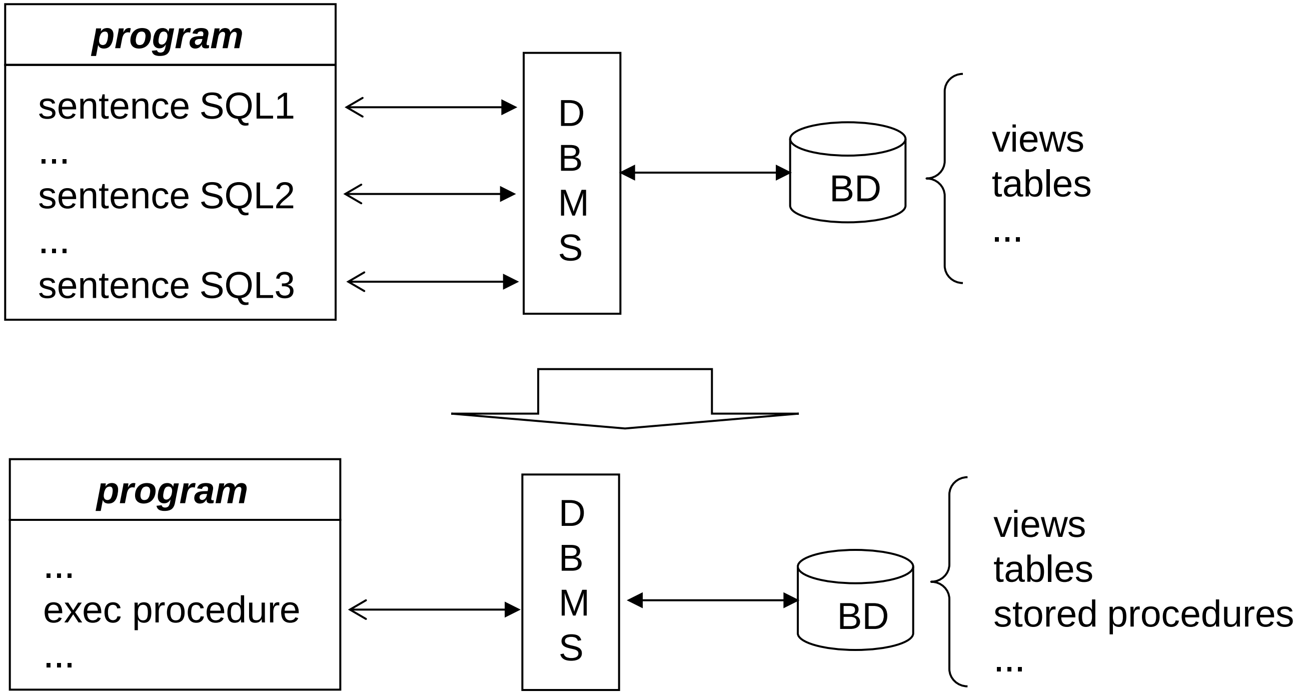
```
CREATE TABLE Invoice (
    member STRING,
    month MONTH,
    payed BOOLEAN,
    amount MONEY,
    PRIMARY KEY (member, month),
    FOREIGN KEY (member) references(Member) ON DELETE CASCADE)
```

When removing a Member, the DBMS automatically removes its invoices

Relational Data Base Technology:

Operation design, stored procedures

- They are used to:
  - Make application development simpler
  - Improve the DB perfomance
  - Control the operations that users execute against the DB
- May compromise portability



Relational Data Base Technology:

Operation design, stored procedures

Member (dni, numM, debt, ba)

Invoice (member, month, payed, amount)

BankAccount (num-ba, balance)

```
CREATE PROCEDURE PayInvoice (dniM, monthTramitation)
RETURNING INTEGER, CHAR(50);
DEFINE error-code INTEGER; ….
ON EXCEPTION SET error-code, error-miss;  RETURN error-code, error-miss END EXCEPTION;

IF ((SELECT COUNT(*) FROM invoice WHERE month=monthTramitation AND member=dniM)=1) THEN
   LET amount, l-payed = (SELECT amount, payed FROM invoice  WHERE month=monthTramitation AND member=dniM));
    IF (‘Y’ = l-payed ) THEN RAISE EXCEPTION 2, ‘The invoice is already payed’;
        ELIF LET balance, numba = (SELECT c.balance,c.num-ba FROM member s, bankaccount c WHERE s.ba=c.num-ba
                                                                                                     and s.dni=dniM);
                IF balance < amount THEN RAISE EXCEPTION 3, ‘The member has not balance enough’;
                ELSE UPDATE invoice SET payed = ‘Y’ WHERE month=monthTramitation AND dni=dniM;
                          UPDATE member s SET s.debt = s.debt-amount WHERE s.dni=dniM;
                          UPDATE bankaccount c SET c.balance = c.balance-balance WHERE num-ba =c.num-ba ENDIF
ELSE RAISE EXCEPTION 1, ‘The member has not invoices this month’; END IF;
RETURN 0,’PayedInvoice’;
END PROCEDURE;
```

References

- M. Fowler
  Patterns of Enterprise Application Architecture
  Addison-Wesley, 2003
- H. Garcia-Molina, J. Ullman, J. Widom
  Database Systems Implementation
  Prentice-Hall, 2000.
- J.Melton, A.Eisenberg
  Understanding SQL and Java Together
  Morgan-Kaufmann, 2000.
- Christian Bauer, Gavin King
  Hibernate in Action
  Manning Publications Company, 2004