UNIQUE DB : view integration

Technologies for Information Systems

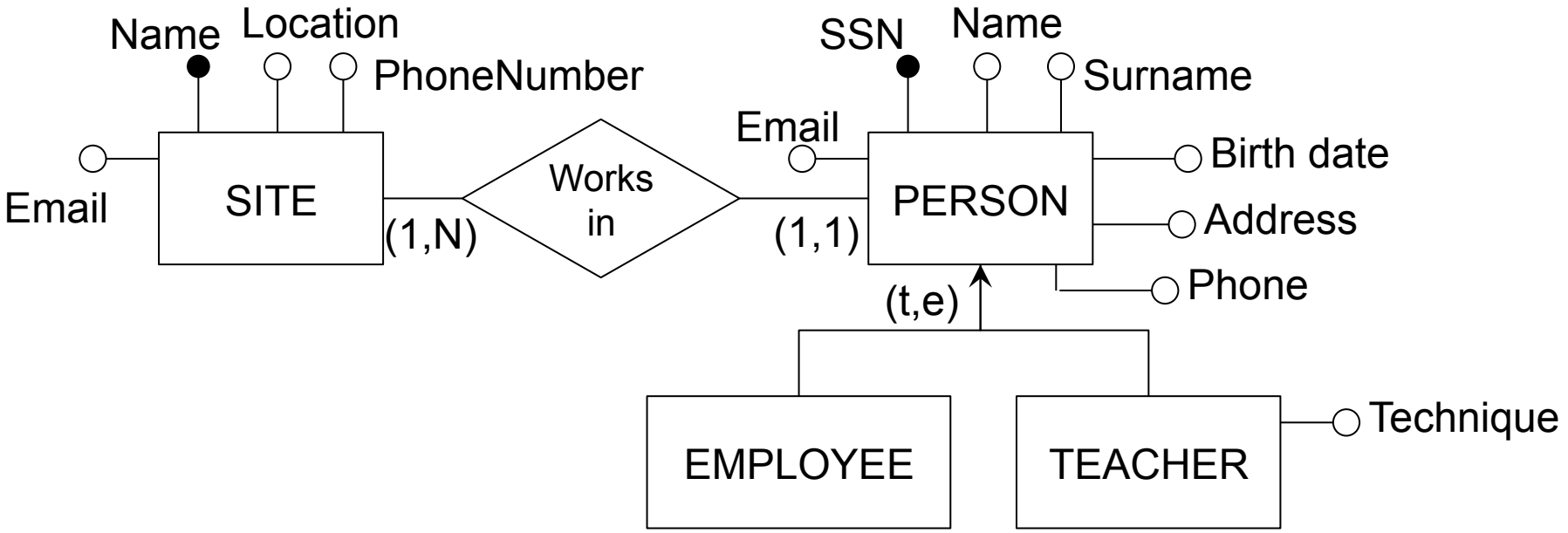
Problem specification

We want to define the database of a ski school having different sites in Italy. Each site has a name, a location, a phone number and an e-mail address. We want to store information about customers, employees and teachers (SSN, name, surname, birth date, address, and phone). For each teacher we store also the technique (cross-country, downhill, snow board). *The personnel office has a view over the personnel data, that is, ski teachers and employees*.

The school organizes courses in the different locations of the school. *The course organization office has a view over these courses.* The courses have a code (unique for each site), the starting date, the day of the week in which the course takes place, the time, and the kind of course (cross-country, downhill, snow board), the level, the number of lessons, the cost, the minimal age for participant. For each course, a unique teacher is associated, and the participants.

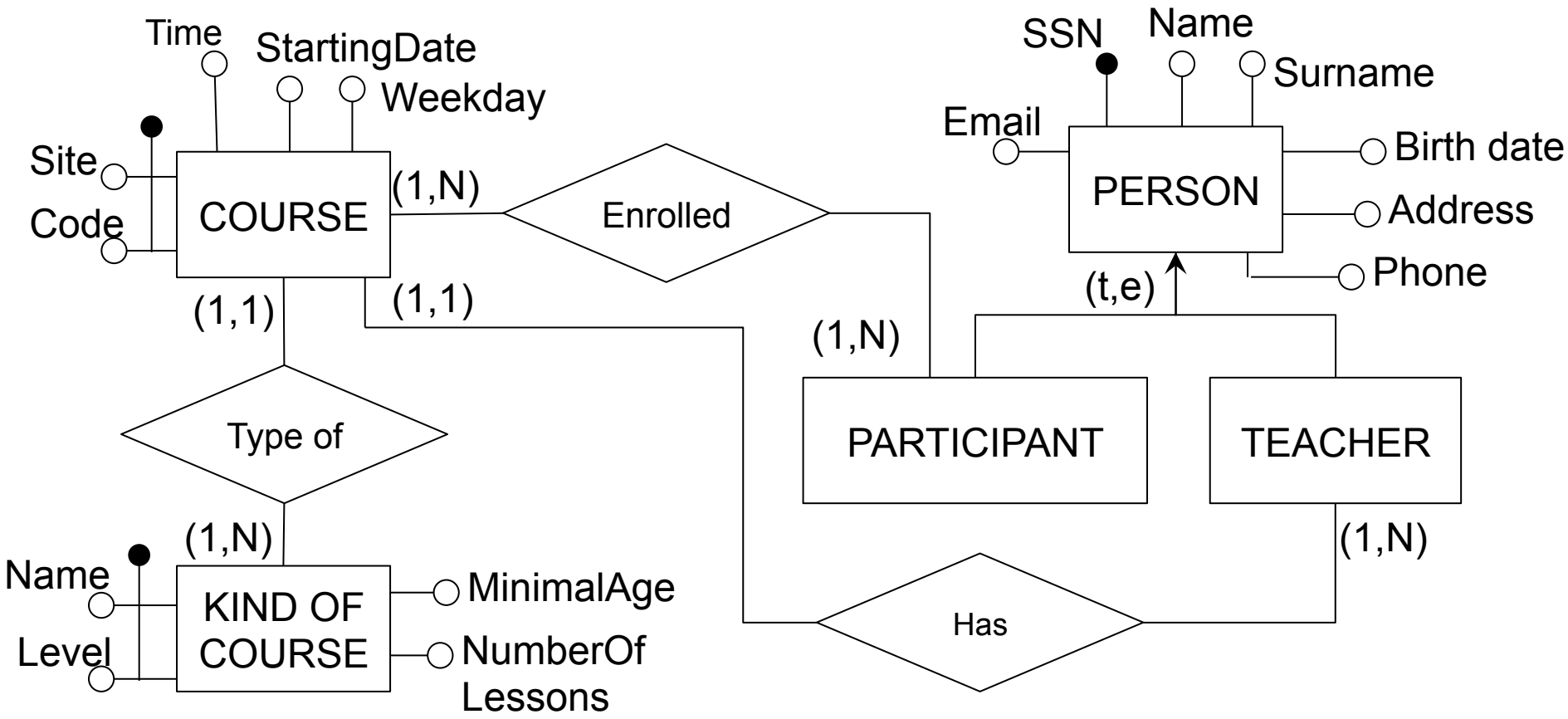
Subschema (view) conceptual design

ER1 – Personnel office

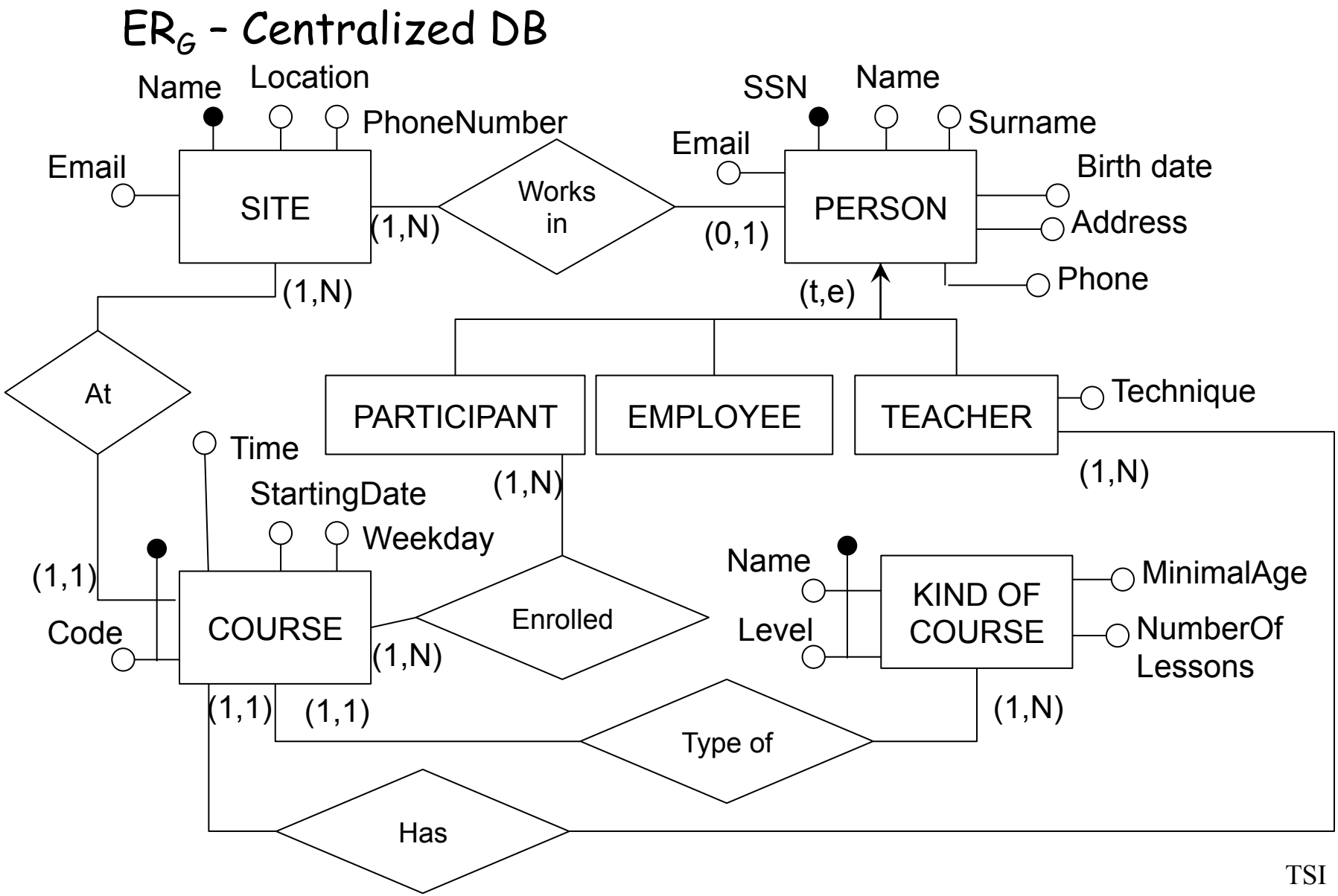


Subschema (view) conceptual design

ER2 – Course organization office



Integration and restructuring



Global logical schema

Logical schema of the integrated database

G.PERSON (SSN, name, surname, birthdate, address, phone, email, technique*, nameSite*, personRole)
personRole can assume the values: Participant, Employee, Teacher

G.SITE (name, location, phoneNumber, email)

G.KIND_OF_COURSE(name, level, minimalAge, numberOfLessons)

G.COURSE(nameSite, code, startingtDate, time, weekday, SSNTeacher, nameKind, levelKind)

G.ENROLLED (SSNParticipant, nameSite, code)

Subsystem logical schema

Logical schema of the personnel office view

L1.PERSON (SSN, name, surname, birthdate, address, phone, email, technique*, nameSite, personRole)
personRole can assume the values: Employee or Teacher

L1.SITE (name, location, phoneNumber, email)

Subsystem logical schema

Logical schema of the course office view

L2.PERSON (SSN, name, surname, birthdate, address, phone, email, personRole)
personRole can assume the values: Participant or Teacher

L2.KIND_OF_COURSE(name, level, minimalAge, numberOfLessons)

L2.COURSE(nameSite, code, startingtDate, time, weekday, teacherSSN, nameKind, levelKind)

L2.ENROLLED (SSNParticipant, nameSite, code)

View definition: Personnel office view

Here we copy the same attributes with no condition:

SQL:

CREATE VIEW **L1.SITE** (name, location, phoneNumber, email) AS
SELECT name, location, phoneNumber, email
FROM G.SITE;

Datalog:

L1.Site (name, location, phoneNumber, email) ← *G.Site (name, location, phoneNumber, email)*

View definition: Personnel office view

SQL:

CREATE VIEW **L1.PERSON** (SSN, name, surname, birthdate, address, phone, email, technique*, nameSite, personRole) AS
SELECT SSN, name, surname, birthdate, address, phone, email, technique, nameSite, personRole
FROM G.PERSON
WHERE G.PERSON.personRole='Teacher' OR
G.PERSON.personRole='Employee';

Datalog:

L1.Person(SSN, name, surname, birthdate, address, phone, email, technique, nameSite, 'Teacher') ← *G.Person (SSN, name, surname, birthdate, address, phone, email, technique, nameSite, 'Teacher')*

L1.Person(SSN, name, surname, birthdate, address, phone, email, technique, nameSite, 'Employee') ← *G.Person (SSN, name, surname, birthdate, address, phone, email, technique, nameSite, 'Employee')*

Note: *this is the way we define the UNION in Datalog !!*

View definition (SQL-based)

Course organization office (part 1)

CREATE VIEW **L2.PERSON** (SSN, name, surname, birthdate, address, phone, email, personRole) AS
SELECT SSN, name, surname, birthdate, address, phone, email
FROM G.PERSON
WHERE G.PERSON.personRole='Teacher' OR
G.PERSON.personRole='Participant';

CREATE VIEW **L2.KIND_OF_COURSE**(name, level, minimalAge, numberOfLessons) AS
SELECT name, level, minimalAge, numberOfLessons
FROM G.KIND_OF_COURSE;

AND Datalog?

View definition (SQL-based)

Course organization office (part 2)

CREATE VIEW **L2.COURSE**(nameSite, code, startingtDate, time, weekday, teacherSSN, nameKind, levelKind) AS
SELECT C.nameSite, C.code, C.startingtDate, C.time, C.weekday, SSNTeacher, nameKind, levelKind
FROM G.COURSE C;

CREATE VIEW **L2.ENROLLED** (SSNParticipant, nameSite, code) AS
SELECT SSNParticipant, nameSite, code
FROM G.ENROLLED;

AND Datalog?

Query processing in a centralized DB

- User issues query Q on the user view → $Q(V_i)$
- Query composition → $Q \circ V_i\ (DB)$
- Answer is in terms of the base relations (global schema) or of the viewed relations (external schema), depending on how sophisticated the system is

Query processing

Find all the participants in courses of levelkind 2 who were born after 1980 issued by an employee of the course office

SELECT SSN, name, surname, birthdate FROM L2.PERSON

WHERE L2.PERSON.personRole='Participant' AND L2.PERSON.birthdate>1980 AND L2.PERSON.SSN=L2.ENROLLED.SSNParticipant AND

L2.ENROLLED.nameSite= L2.COURSE.nameSite AND L2.ENROLLED.code=L2.COURSE.code AND L2.COURSE.levelkind=2

see how easy it is to write it in datalog!!

Young-level-2(SSN, name, surname, birthdate) ← L2.PERSON (SSN, name, surname, birthdate, _, _,_,'Participant') , birthdate>1980, ENROLLED (SSN, nameSite, code), L2.COURSE(nameSite, code, _, _, _, _, _, 2).

?- Young-level-2(SSN, name, surname, birthdate) .

Note the use of the "don't care symbol"

Query processing on the global DB

Find all the participants in courses of levelkind 2 who were born after 1980 (issued by an employee of the Course Office)

Young-level-2(SSN, name, surname, birthdate) ← L2.PERSON (SSN, name, surname, birthdate, _, _, _, 'Participant') , birthdate>1980, L2.ENROLLED (SSN, nameSite, code), L2.COURSE(nameSite, code, _, _, _, _, _, 2).

Becomes:

Young-level-2(SSN, name, surname, birthdate) ← G.PERSON (SSN, name, surname, birthdate, _, _, _,_,_, 'Participant') , birthdate>1980, G.ENROLLED (SSN, nameSite, code), G.COURSE(nameSite, code, _, _, _, _, _, 2).

Note: G.Person has more attributes than L2.Person!