

Answers

to odd numbered exercise questions

Terry Halpin and Tony Morgan
Neumont University

Answers in this document are supplied for odd numbered exercise questions, as well as a few even numbered questions. Answers to the other exercise questions are available as a separate document to class instructors using the book as a set text.

Exercise 2.1

1. (a) C (b) A (c) B

Exercise 2.2

1. (a) Ensure database updates are consistent with the conceptual schema. Answer queries about the UoD.
(b) Declarations of: fact types; constraints; derivation rules.
(c) False. With compound transactions, only the collective effect of all the elementary updates is considered.
3. (a) accepted
(b) rejected. C1 violated. Sue must be enrolled in a degree.
(c) accepted
(d) rejected. C1 violated. Fred may not enroll in two degrees.
(e) rejected. Fact type not recognized.
(f) single, married, widowed, divorced
(g) accepted
(h) rejected. C2 violated.
(i) accepted.
(j) accepted.
(k) rejected. C4 violated. Transition single to divorced illegal.
(l) No
(m) Fred, Bob
(n) Sue
(o) rejected. Fact type not recognized. -- Such information would typically be derived

Final state of database: Student 'Fred' is enrolled in Degree 'BSc'.
Student 'Sue' is enrolled in Degree 'MA'.
Student 'Bob' is enrolled in Degree 'BSc'.
Student 'Bob' has MaritalState 'single'.
Student 'Sue' has MaritalState 'married'.

Information Modeling and Relational Databases: Answers (odd)–2

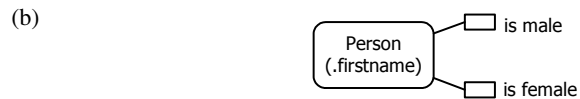
5. (a) **begin**
 add: Employee 'Adams' works for Department 'Health'
 add: Employee 'Adams' speaks Language 'English'
 end
 begin
 add: Employee 'Brown' works for Department 'Health'
 add: Employee 'Brown' speaks Language 'English'
 end
Brown's data may be entered first. Within a compound transaction the order is irrelevant.
- (b) (i) Employee 'Adams' works for Department 'Health'. -- language missing
 (ii) Employee 'Adams' works for Department 'Health'.
 Employee 'Adams' speaks Language 'English'.
 Employee 'Adams' works for Department 'Education'. -- cannot work for 2 depts

Exercise 3.3

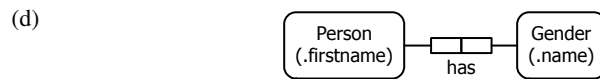
1. (a), (f), (j)
- 3–4. These questions reveal that columns with the same name need not have the same meaning. In the absence of helpful table names, we make an educated guess. For example:
3. Athlete (.name) 'Jones EM' has height of Length (cm:) 166.
 aliter: Athlete (.name) 'Jones EM' has Height 166 (cm: Length).
4. Athlete (.name) 'Jones EM' pole vaults a height of Length (cm:) 400.
 aliter: Athlete (.name) 'Jones EM' pole vaults Height 400 (cm: Length).
5. The uniqueness of person entries suggests that each row contains two elementary facts. For example, the top row may be verbalized as:
- Person (.name) 'Jones EM' has height of Length (cm:) 166.
 Person (.name) 'Jones EM' was born in Year (CE) 1955.
- However, this is open to interpretation, as the population might not be significant (only a domain expert can know for sure what is intended). So it is still possible that each row contains just a ternary, elementary fact, e.g.,
- Person (.name) 'Jones EM' had a height of Length (cm:) 166 in Year (CE) 1995.
7. Person (.surname) 'Codd' is an internal member of Panel (.name) 'Databases'.
 Person (.surname) 'Tenshtein' is an external member of Panel (.name) 'Databases'.
9. For all relevant answers, we use "firstname" to mean "first given name" (to avoid confusion with languages such as Japanese where the first name listed is normally the family name). Such a definition or description may be noted in an ORM modeling tool.
- Person (.firstname) 'Ann' is a parent of Person (.firstname) 'Colin'.
 Person (.firstname) 'Ann' is a parent of Person (.firstname) 'David'.
 Person (.firstname) 'Ann' is a parent of Person (.firstname) 'Eve'.
 Person (.firstname) 'Bill' is a parent of Person (.firstname) 'Colin'.
 Person (.firstname) 'Bill' is a parent of Person (.firstname) 'David'.
 Person (.firstname) 'Bill' is a parent of Person (.firstname) 'Eve'.
11. The Fruit named 'Apple' is harvested in the Country named 'Australia' in the Month named 'June'. etc.

Exercise 3.4

1. (a) Reference schemes: Person (.firstname).
Facts: Person 'Fred' is male. Person 'Ann' is female.



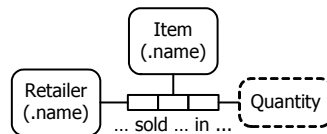
- (c) Reference schemes: Person (.firstname); Gender (.name)
Facts: Person 'Fred' has Gender 'male'. Person 'Ann' has Gender 'female'.



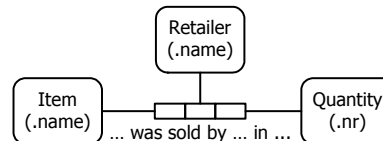
3. Only the schema diagrams (two of six possible versions based on reading order) are shown here. You should also verbalize and populate.



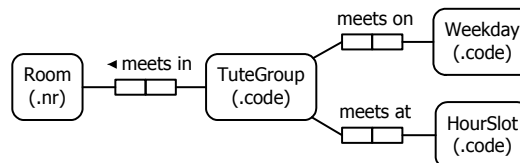
If you wish to think of Quantity as a pure number, then you may verbalize and model it as a value type as shown below. Some ORM modelers treat this as an error, since they view even pure numbers (as distinct from numerals) as entities, not values. So whether this is allowed or not depends on the precise definition of what a value is. For mapping purposes, it makes so difference, so we allow this relaxation.



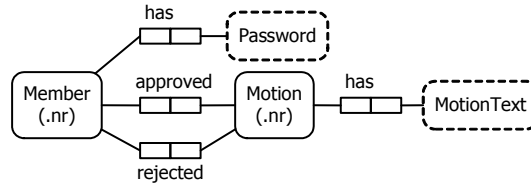
4. This schema is equivalent to the previous ones, which may be used again instead. Verbalization and population are omitted here.



- 5.

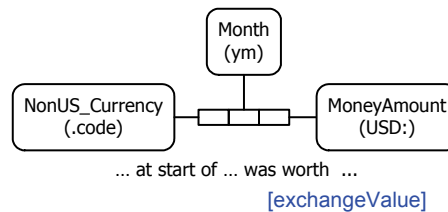


7.

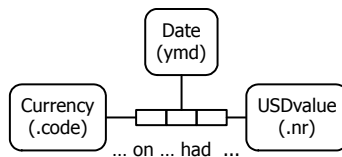


8. (a) Illegal. Objects must play a role. Directly connecting object types is forbidden.
 (b) Illegal. A role can be played by at most one object type.
 (c) Illegal. Inside role has no object type.
 (d) OK. For example: Person has Gender; Person was born in Country.
 (e) Illegal. A role cannot be played by more than one object type.
 (f) OK. For example: Person introduces Person to Person.
 (g) OK, e.g.: Company employs Person; Company buys Computer; Person uses Computer.
 (h) OK, e.g.: Person gives Present to Person on Date.
 (i) Illegal. Object types should not be included in the nesting.
 (j) OK, e.g.: Person is a parent of Person; Person has IQ.
 (k) OK, e.g.: Person ordered Product: objectify as “Order”.
 Order was issued on Date. Order was paid on Date.
 (l) Illegal. An extended form of the error in (e).

9. Here is one solution that restricts the valuation date to the start of a month. A restriction to specific non-US currencies (e.g. AUD and XEU) may be added as a value constraint (see Chapter 6).

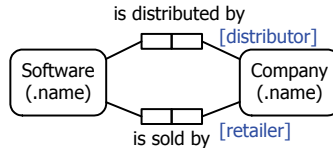


Here is another solution that allows any currency for the first role (USD would then yield an exchange value of 1) and any date for the valuation. It also avoids using USD as the standard monetary unit. As you can see, there can be many correct solutions, so long as the domain experts agree on how the terms used are to be interpreted.



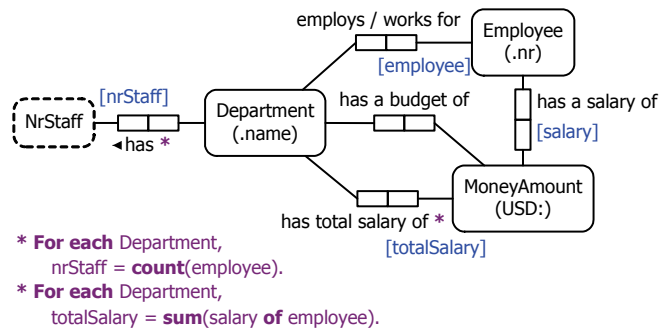
Exercise 3.5

1.

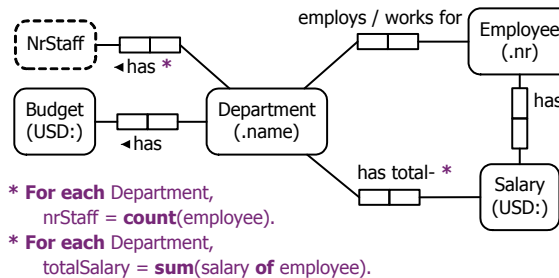


This solution assumes that distributors and retailers are companies. It also uses “sold” in the sense of retailed (e.g., rather than wholesaled).

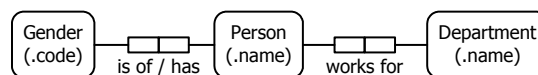
3.



Note: “employee.salary” may be used instead of “salary of employee” At the time of writing, the NORMA tool is being extended to automatically generate implicit role names using object type names (where the object type plays no other role in the relevant fact type) and concatenated names obtained from hyphen binding. For example, using unit-based reference mode support (where the USD unit is based on the Money dimension), the following alternative solution is then allowed without explicitly adding role names.

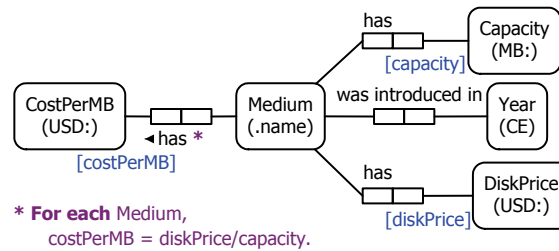


5.



aliter: Person is a female employee in Department;
Person is a male employee in Department

7.



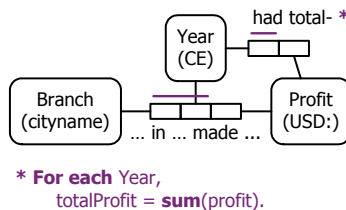
Note: The role names shown above may be assumed once NORMA supports implicit role names. Where relevant, later answers will typically omit implicit role names.

Exercise 4.2

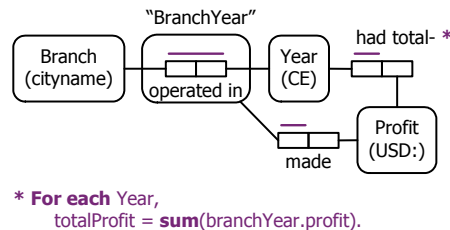
- Uniqueness constraints apply to: (a) B column; (b) AB combination; (c) A column and B column; (d) A column.
- Uniqueness constraints apply to: (a) each role (implied); (b) left role; (c) 3.3.3 role of Athlete, 3.3.4 role of Athlete, 3.3.5 each role of Person, 3.3.6 combination of Person and Year roles, 3.3.7 both fact types are many:many (population is not significant), 3.3.8 combination of Student and Course roles, 3.3.9 many:many, 3.3.10 both fact types are many:many; (d) each of the three roles of TuteGroup (later we see how to add external uniqueness constraints, e.g. the combination of Room, Day and Hour is probably unique); (e) each role of Project, and the birth and salary roles of Person; (f) each role of Person, and the budget role of Department.

Exercise 4.3

- (a) AB, AC (b) AB, AC, BC (c) ABC (d) AC
- (a)



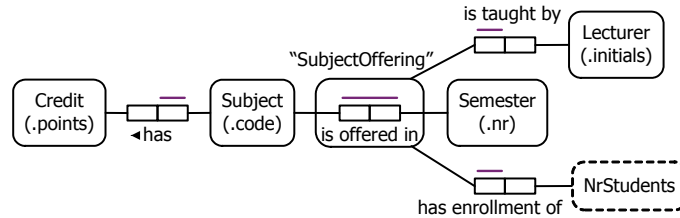
(b)



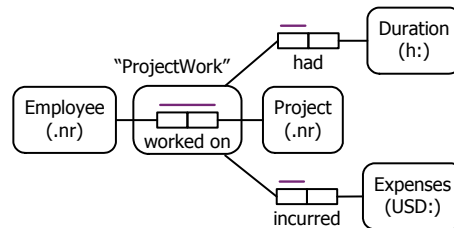
Note: “profit of branchyear” may be used instead of “branchYear.profit”.

Exercise 4.4

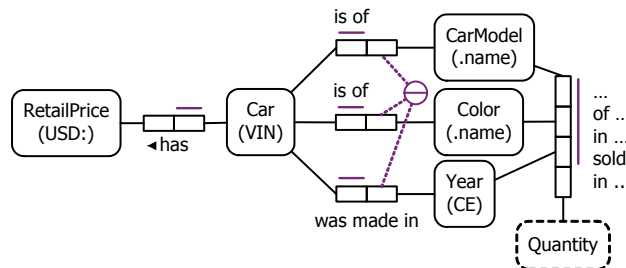
1. (a)



(b)



3.

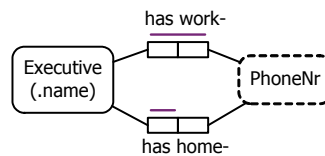


This schema makes no attempt to record history about specific cars formerly in stock.

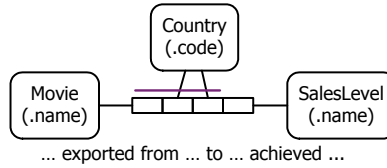
Exercise 4.5

1. (a), (b), (d)

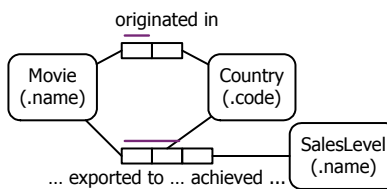
3. Since each executive has only one home phone, this information may be split off without information loss. Hence the proposed ternary is compound, and must be split into two binaries, as shown below.



5.



This is unacceptable, since it cannot store the data in the final row. We must be able to store facts about movie origins without knowing their export figures. The quaternary should be split, as shown below:



Exercise 4.6

1. (a) Constraints on AC, BC.

(b) Projections:

a1	c1
a1	c2
a2	c1
a2	c2

b1	c1
b1	c2
b2	c1
b2	c2

Join:

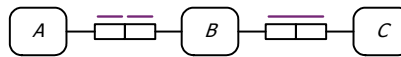
a1	b1	c1
a1	b2	c1
etc.

× so unsplittable

(c) AB projection: a1 b1 BC projection as in (b)
a2 b2

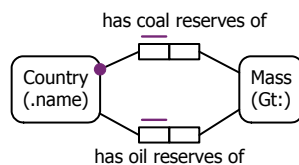
The join gives the original table. So, assuming significant data, it is splittable in this way.

(d)

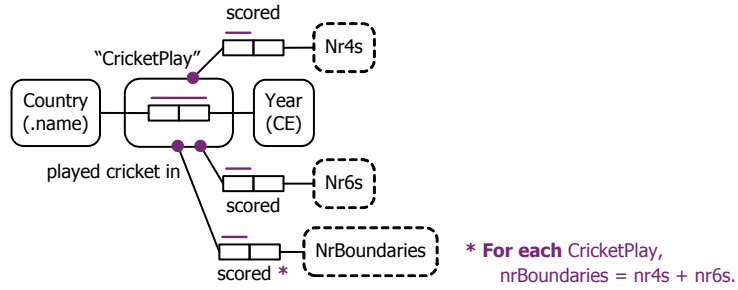


Exercise 5.2

1.

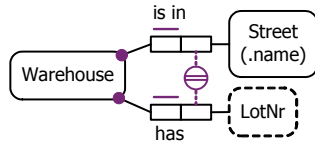


3.

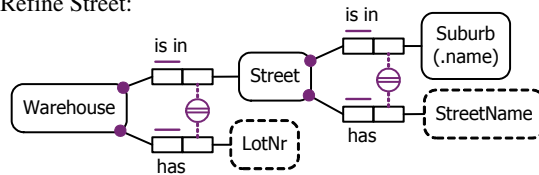


Exercise 5.3

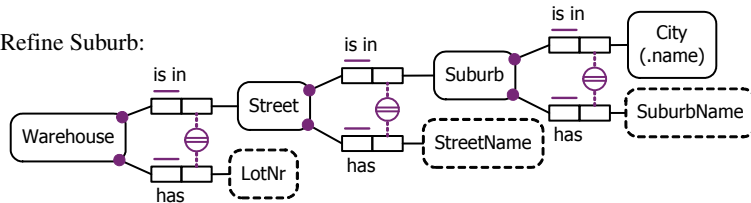
1. (a)



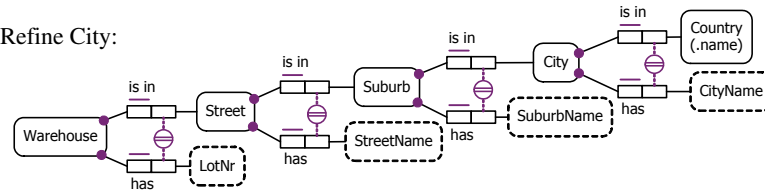
(b) Refine Street:



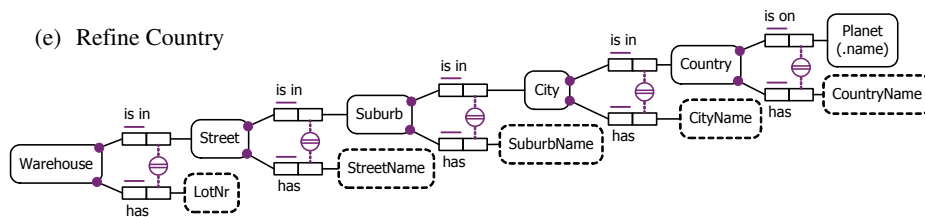
(c) Refine Suburb:



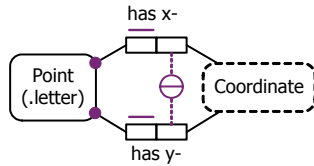
(d) Refine City:



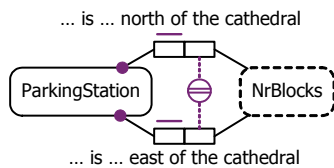
(e) Refine Country



3. (a)

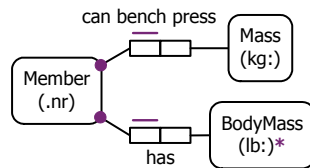


(b)

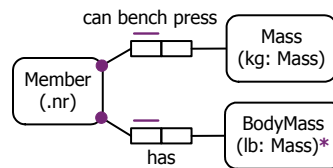


Note: This schema fragment is just a reference scheme for parking stations. We may now add facts about them (e.g. hours open). South and West coordinates have negative values.

5.



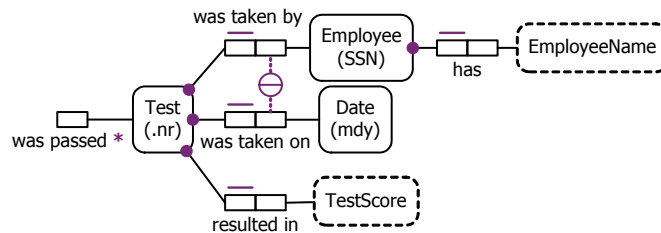
* 1 lb = 0.454 kg



* 1 lb = 0.454 kg

Note: This schema uses kg as the standard unit of mass. The bench press fact type assumes earth surface gravity. If you choose lb as the standard unit, the conversion rule is * 1 kg = 2.205 lb. If you choose the term “weight”, either add definitions to indicate that strictly you mean mass, or change the units to lb wt and kg wt.

7.

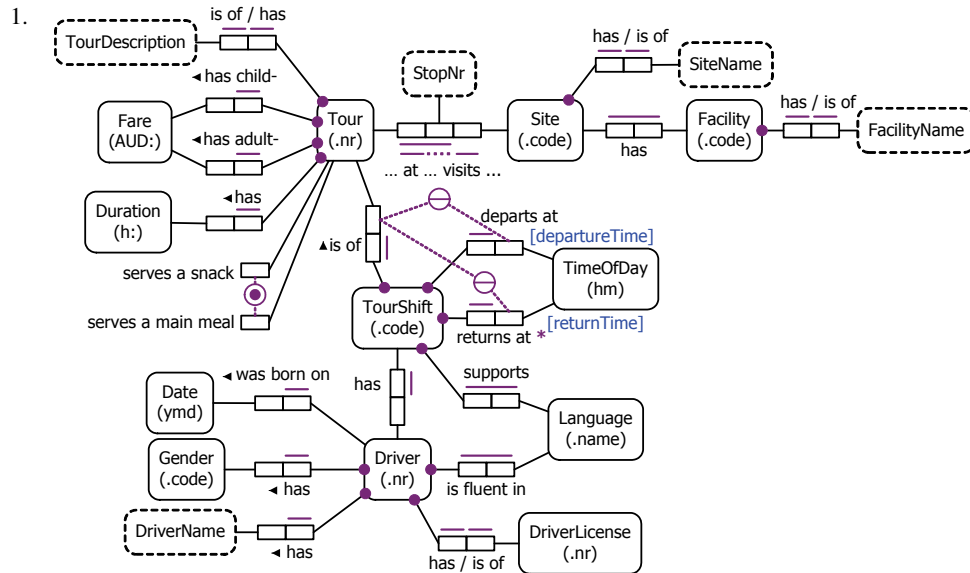


* Test was passed **iff**
Test resulted in a TestScore >= 80.

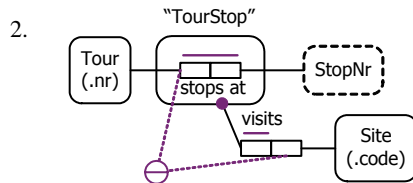
Note: This derivation rule is neater in relational style, as shown above, but here is one way to specify it in attribute style:

* **For each** Test,
was passed = **true iff** testScore >= 80.

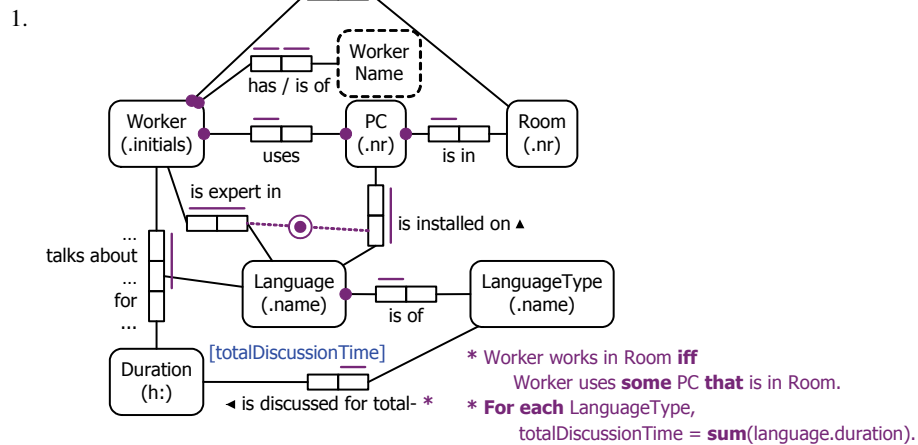
Exercise 5.4



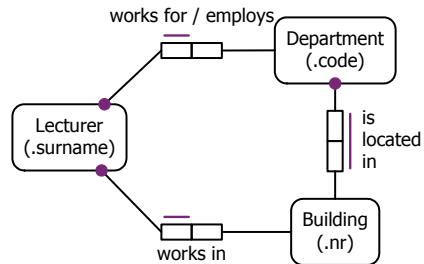
* For each TourShift,
returnTime = departureTime + tour.duration.



Exercise 5.5



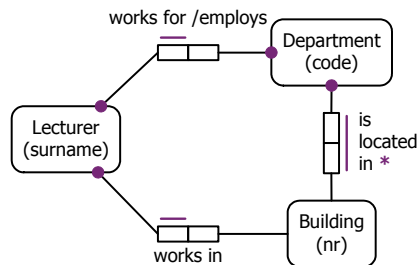
2.



None of the fact types is (fully) derivable, so an asserted fact type is needed for each. The department location fact type is *semiderivable* if the following **if**-rule (not **iff**) holds:

⁺Department is located in Building **if**
 Department employs **some** Lecturer **who** works in Building.

3.

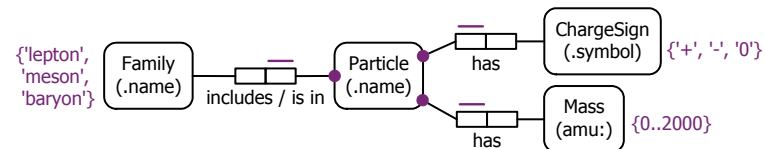


-- Assuming each department employs lecturers in all its buildings
 * Department is located in Building **iff**
 Department employs **some** Lecturer **who** works in Building.

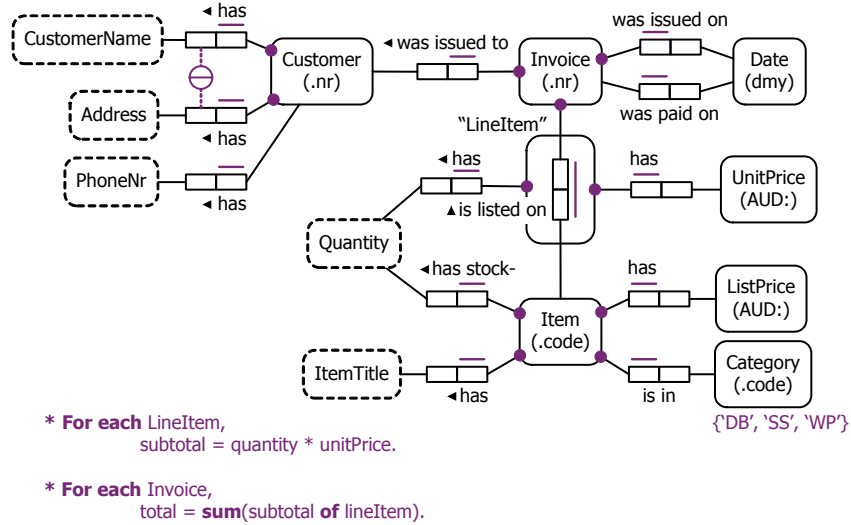
This solution assumes that for each department, we must record at least one lecturer for each of the department's buildings. If this is not true, then none of the fact types is fully derivable, and our solution is the same as for Question 2 except for the extra mandatory constraint.

Exercise 6.3

1.

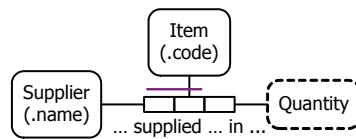


3.

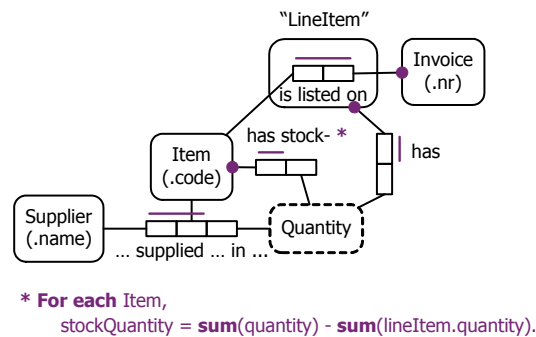


Note: The derived fact types LineItem has Subtotal(AUD:) and Invoice has Total(AUD:) may be displayed on the diagram if desired.

5. (a)



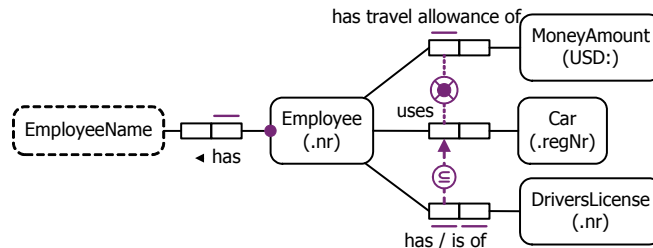
(b) (i)



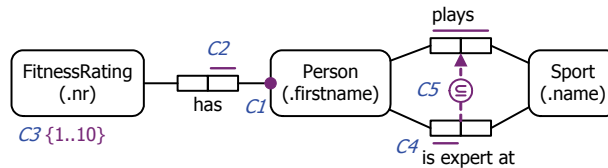
- (ii) It is not practical to use this rule, since items may be stolen or misplaced. So periodic stocktakes must be taken to determine the exact number in stock, and stocktake numbers should be stored. Such a rule may be used however to provide default stock figures which may be compared with stocktake figures to discover discrepancies. In such a case, a more complex rule is needed to take into account previous discrepancies as well as stock write-offs (e.g. due to damage) and items that are returned from the buyer.

Exercise 6.4

1.

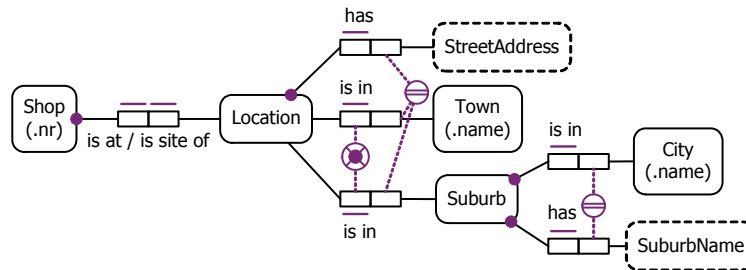


3.



Note: If Sport plays no other roles, there is an implied mandatory role constraint on the righthand role of Person plays Sport. Why?

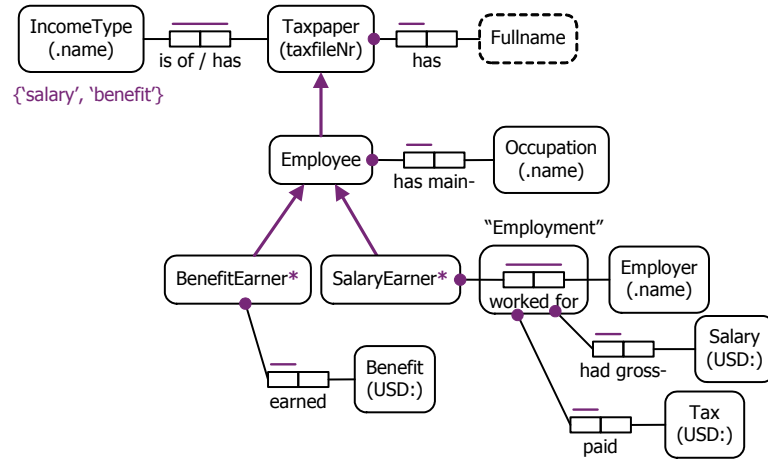
5.



Exercise 6.5

1. (a) (i) ORM requires subtypes to be proper, and types (as distinct from their populations) to be nonempty. For example: $B = \{1, 2\}$ $C = \{1\}$.
 (ii) One example is: $B = \{1, 2\}$ $C = \{2, 3\}$ $D = \{2\}$.
- (b) (i) arcs must be directed (is A a subtype of B or vice versa?)
 (ii) graph must be acyclic (*proper* subtype)
 (iii) delete arc from C to A (transitively implied)
 (iv) graph must be acyclic (*proper* subtype)
 (v) primitive types A and B must be exclusive, so they cannot have a common subtype C

3. (a)



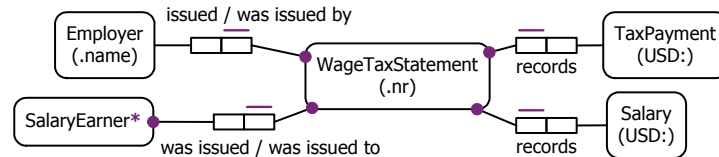
***Each Employee is a Taxpayer who has some IncomeType.**

***Each BenefitEarner is a Taxpayer who has IncomeType 'salary'.**

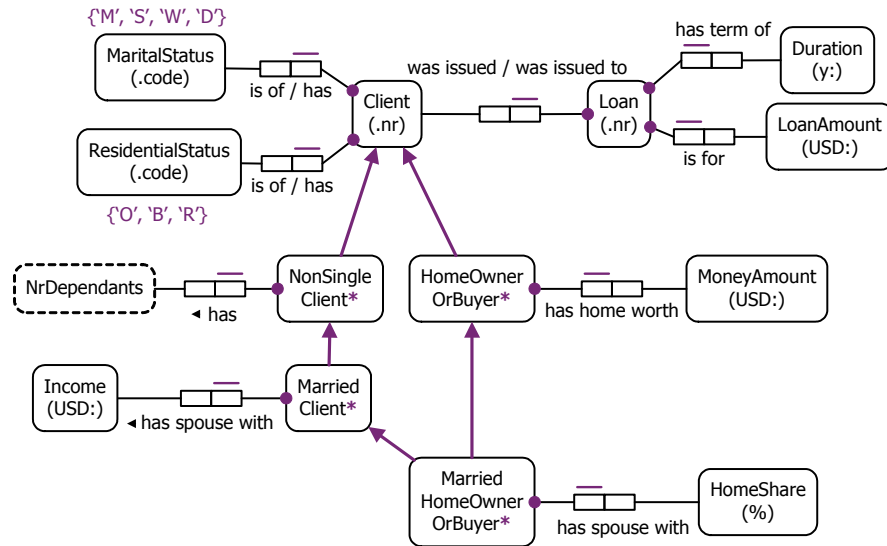
***Each SalaryEarner is a Taxpayer who has IncomeType 'benefit'.**

Notes: For simplicity, this solution ignores the need to identify employers as taxpayers. If a salary earner pays no tax we assume this is explicitly recorded as a zero amount. For this example (and in practice many others) explanatory notes provide a useful supplement to the basic conceptual schema. In this case, the term “Taxpayer” should be clarified as meaning “potential payer of tax”, i.e. anybody who has a tax file number. Whether a taxpayer actually pays tax depends on other factors (e.g. the taxpayer’s income).

(b)



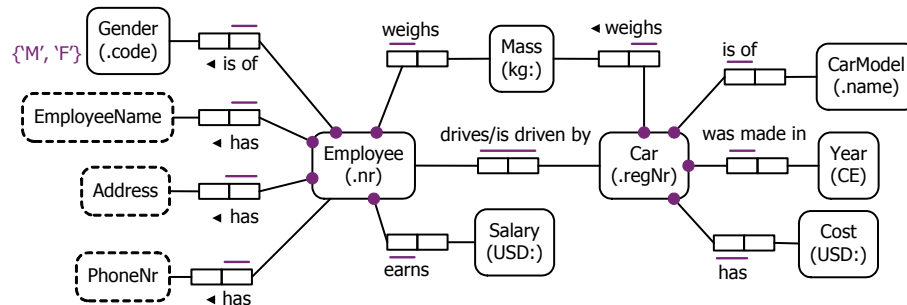
5.



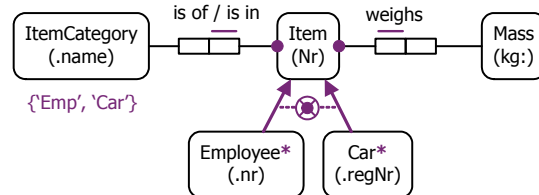
- *Each NonSingleClient is a Client who has MaritalStatus <> 'S'.
- *Each HomeOwnerOrBuyer is a Client who has ResidentialStatus in {'O', 'B'}.
- *Each MarriedClient is a Client who has MaritalStatus 'M'.
- *Each MarriedHomeOwnerOrBuyer is a MarriedClient and a HomeOwnerOrBuyer.

Exercise 6.6

1. (a)



(b)

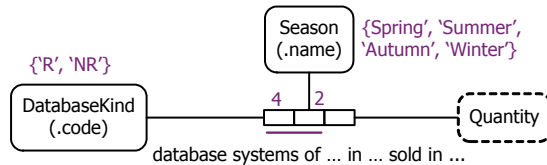


- *Each Employee is an Item that is in ItemCategory 'Emp'.
- *Each Car is an Item that is in ItemCategory 'Car'.

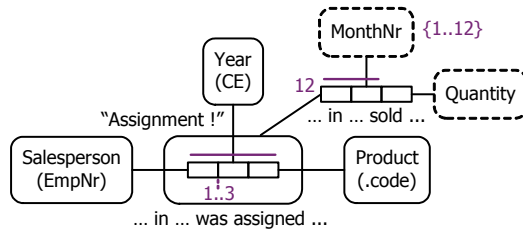
1. (c) Schema (a) is preferable. Employee and Car are exclusive, with different contextual identification schemes, and it is unlikely we would want to list them both in the same column of an output report.

Exercise 7.2

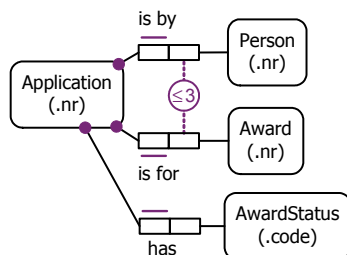
1.



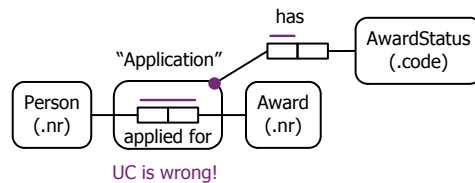
3.



5. (a)



(b)



The spanning uniqueness constraint on the objectified association entails that the same person applied at most once for the same award, but the same person may apply up to 3 times for the same award.

Exercise 7.3

1. (a) R, S, T (b) iR (c) iR, S (d) iR, aS, T (e) iR, aS, T (f) R, T (g) iR, S

Note that (g) is not intransitive (e.g. locations at vertices of a triangle).

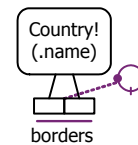
Information Modeling and Relational Databases: Answers (odd)–18

3. Proof is by *reductio ad absurdum*, using a deduction tree. The justification column abbreviations are: P = Premise, NC = Negated Conclusion, df = definition, QN = Quantifier Negation, EI = Existential Instantiation, UI = Universal Instantiation, PC = Propositional Calculus move, AA = Affirming the Antecedent (i.e. *modus ponens*), DC = Denying the Consequent (i.e. *modus tollens*).

- (a) 1. $\forall x \sim xRx$ P (df Irreflex)
 2. $\forall xyz (xRy \ \& \ yRz \rightarrow xRz)$ P (df Trans)
 ✓3. $\sim \forall xy (xRy \rightarrow \sim yRx)$ NC (negated Asym)
 ✓4. $\exists xy \sim (xRy \rightarrow \sim yRx)$ 3 QN
 ✓5. $\sim (aRb \rightarrow \sim bRa)$ 4 EI
 6. aRb 5 PC
 7. bRa 5 PC
 ✓8. $aRb \ \& \ bRa \rightarrow aRa$ 2 UI
 9. aRa 6,7,8 AA
 10. $\sim aRa$ 1 UI
 X 9, 10
- (b) 1. $\forall xy (xRy \rightarrow yRx)$ P (df Sym)
 2. $\forall xyz (xRy \ \& \ yRz \rightarrow xRz)$ P (df Trans)
 ✓3. $\sim \forall xy (xRy \vee yRx \rightarrow xRx)$ NC (negated PopReflex)
 ✓4. $\exists xy \sim (xRy \vee yRx \rightarrow xRx)$ 3 QN
 ✓5. $\sim (aRb \vee bRa \rightarrow aRa)$ 4 EI
 ✓6. $aRb \vee bRa$ 5 PC
 7. $\sim aRa$ 5 PC
 8. $aRb \rightarrow bRa$ 1 UI
 9. $bRa \rightarrow aRb$ 1 UI
 ✓10. $aRb \ \& \ bRa \rightarrow aRa$ 2 UI
 ✓11. $\sim (aRb \ \& \ bRa)$ 10,7 DC
 / \
 12. $aRb \ bRa$ 6 PC
 13. $bRa \ aRb$ 8, 12a AA; 9,12b AA
 14. $aRa \ aRa$ 10,12,13 AA
 X **X**

5. The schema is shown with sample data. If storage is not a problem, use a symmetric relation like this (symmetry can be enforced at update to simplify data entry).

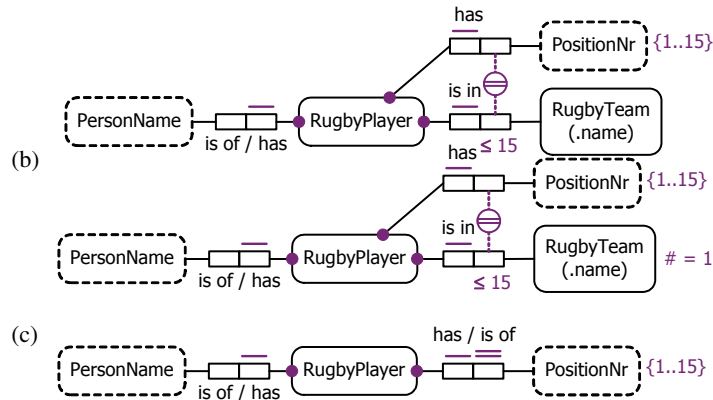
If storage is a problem, use an asymmetric relation (add asymmetric constraint), say *baseBorders* (base table for *borders*), and add the derivation rule: $\text{Country}_1 \text{ borders Country}_2 \text{ iff Country}_1 \text{ baseBorders Country}_2 \text{ or Country}_2 \text{ baseBorders Country}_1$.



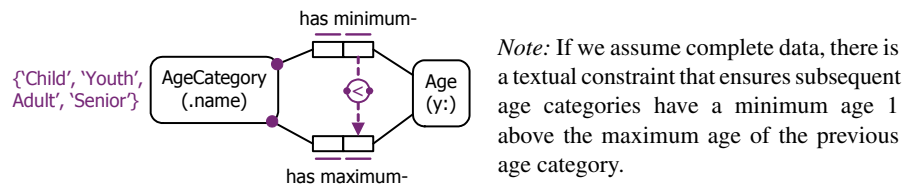
Belgium	France
Belgium	Germany
Belgium	Luxembourg
Belgium	Netherlands
France	Belgium
...	...

Exercise 7.4

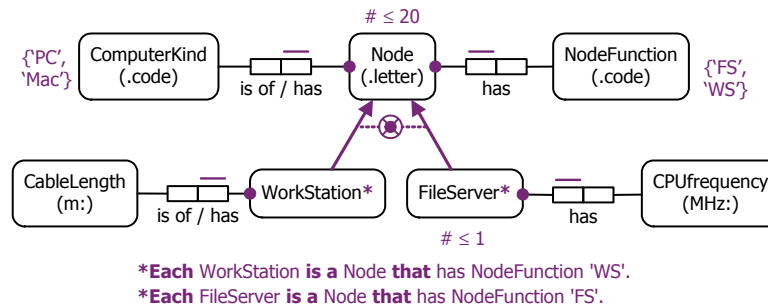
1. (a)



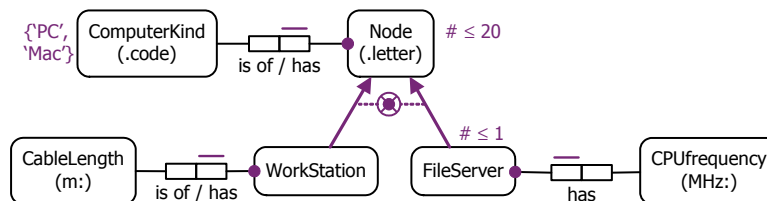
3.



5.



Aliter, simply assert the subtypes in stead of deriving them:



7. The schema is as earlier, but with the following derivation rules added:

Part₁ contains Part₂ **if** Part₁ directly contains Part₂

Part₁ contains Part₂ **if** Part₁ directly contains **some** Part₃ **that** contains Part₂

Exercise 7.5

1. Regarding the left hand fact types:

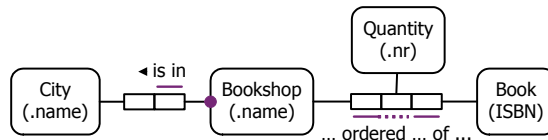
If A 's role in S is populated with a , say, then a must also play A 's role in R (since the pair-subset constraint implies subset constraints on corresponding roles). But this violates the exclusion constraint. Hence the constraint pattern is population-inconsistent.

Regarding the right hand fact types:

If S is meant to identify a typed predicate predicate, then replace it by another symbol to avoid conflation with the other S predicate.

The combination of the subset constraint and the simple UC implies a simple UC on the left role of T .

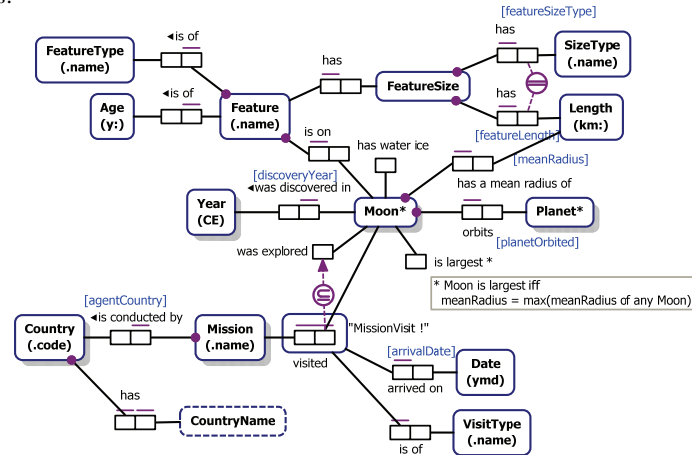
3. The supplied predicate was supposed to have the reading "... located in ... ordered ... of ...". Assuming the population is significant in the sense that each bookshop is located in only one city, the schema should be split into two fact types as shown below:



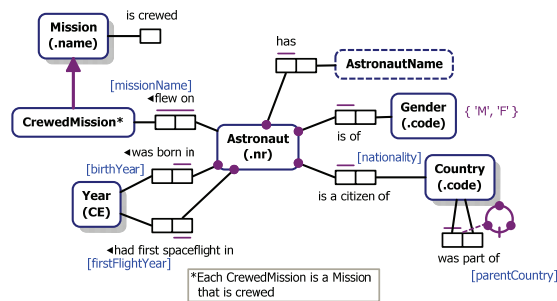
5. The *grandparent_of* relation is semiderived (partly stored and partly derived). This allows storing of the fact that David is a grandparent of Chris without knowing who David's children are. However, suppose we now attempt to store the fact that Ann is a grandparent of Chris. If this fact is accepted, we have derived redundancy since this fact can be derived from the stored facts that Ann is a parent of Bob, and Bob is a parent of Chris, given the derivation rule. This kind of derived redundancy is typically harmless, but if such redundancy is not desired, special exclusion and update constraints may be used.

7. This solution was produced with the NORMA tool, using 3 pages named as shown below.

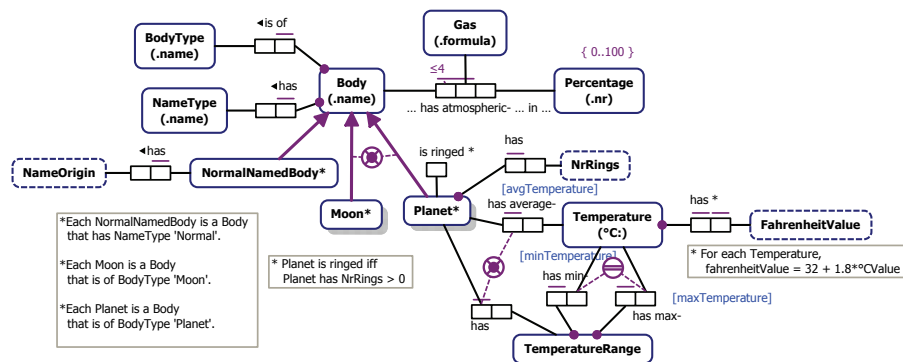
Moon Missions:



Astronauts:

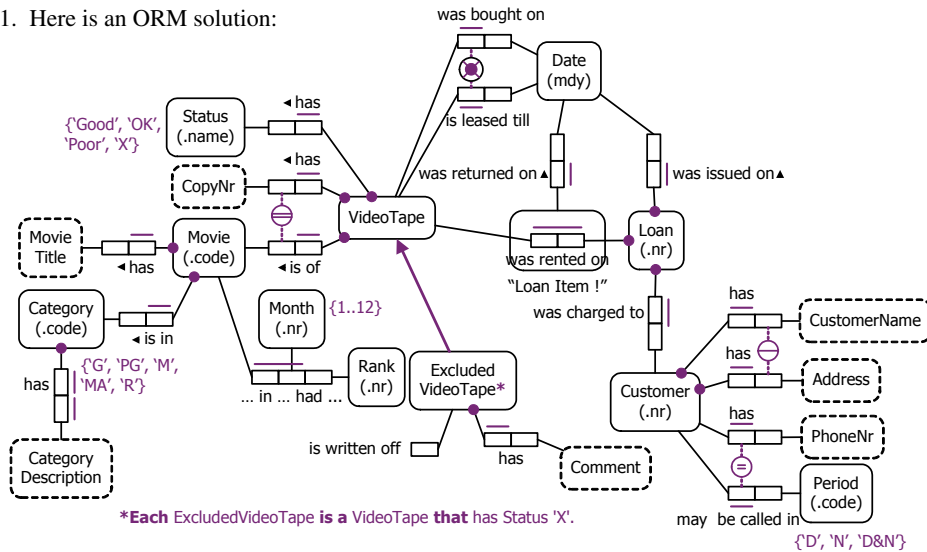


Moons&Planets:

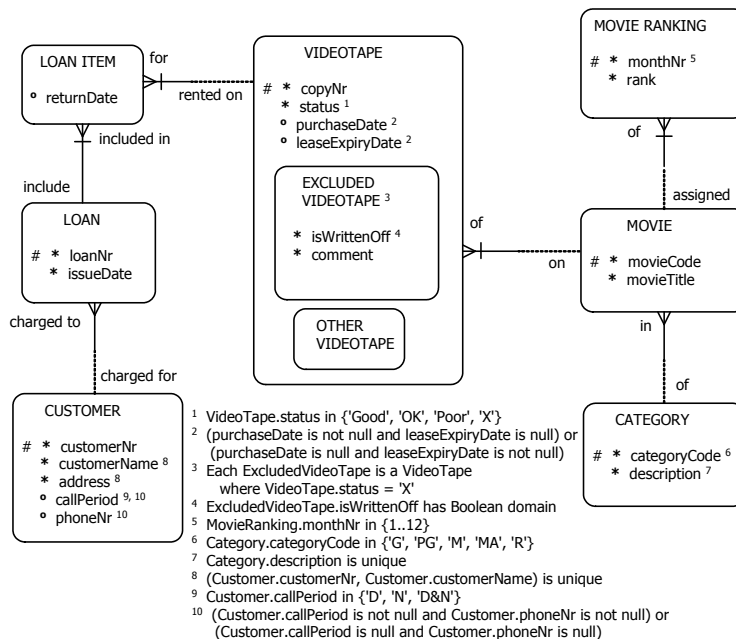


Exercise 8.5

1. Here is an ORM solution:

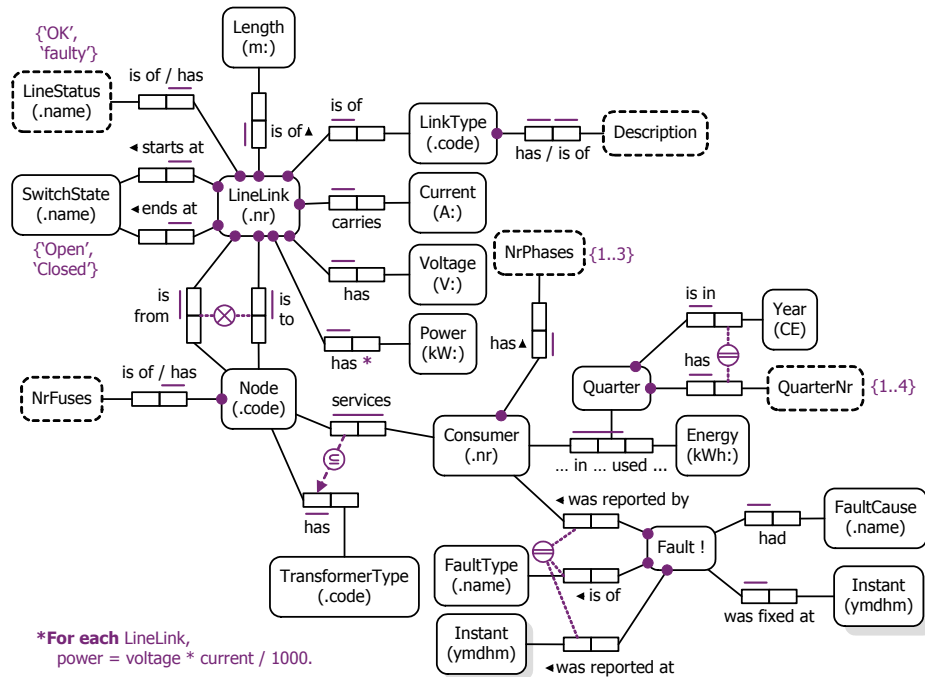


Here is the solution in Barker ER notation, using superscripts to annotate model elements with textual constraints not expressible in the graphic notation. As an additional exercise, suppose that no ties are allowed for movie rankings (so each monthNr, rank pair is assigned to at most one Movie). Add this constraint to both the ORM and the Barker ER solutions. (Hint: you can do this graphically in ORM but not in Barker ER).

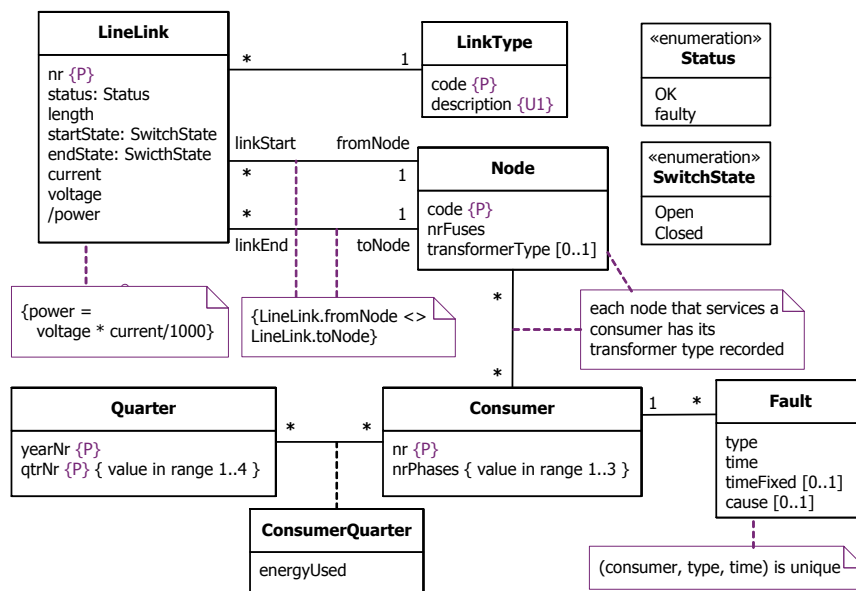


Exercise 9.8

1. (a) Here is an ORM solution:



Here is a UML solution.



1. (b) Introduce a simple identifier for fault (e.g. FaultId). This is advisable even if we model Fault directly as shown in the previous solutions rather than as an objectified association.

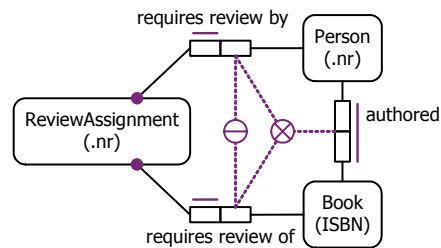
Exercise 10.1

1. (a)

<i>Employment_A</i>				<i>Employment_B</i>			
<i>enr</i>	<i>pnr</i>	<i>startdate</i>	<i>enddate</i>	<i>enr</i>	<i>pnr</i>	<i>startdate</i>	<i>enddate</i>
e1	p1	d1	d3	e1	p1	d1	d4
e2	p2	d2	?	e2	p1	d2	?
e3	p1	d2	?	e3	p1	d3	?

- (b) outer join semantics for UC on (pnr, enddate), i.e. UC applies to outer join
 - (c) inner join semantics, i.e. UC applies to inner join
 - (d) **check(not exists**
 (select pnr, enddate from Employment_A
 group by pnr, enddate
 having count(*) > 1))
 - (e) **check(not exists**
 (select pnr, enddate from Employment_B
 where enddate is not null
 group by pnr, enddate
 having count(*) > 1))

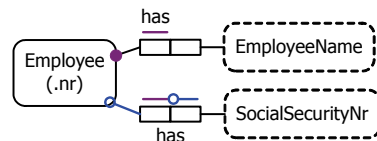
- 3.



Exercise 10.2

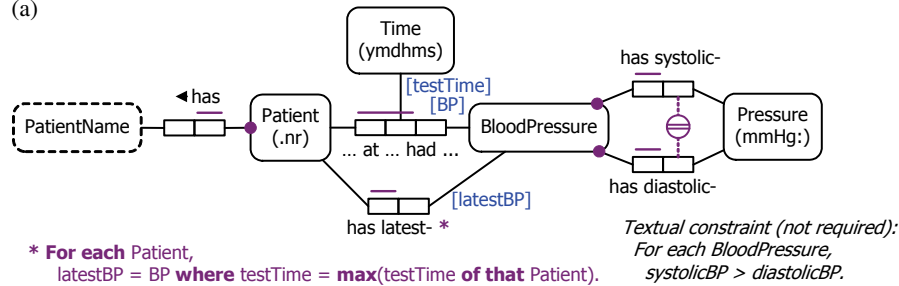
1. (a) No (b) Yes (c) Yes (d) Yes

- 3.

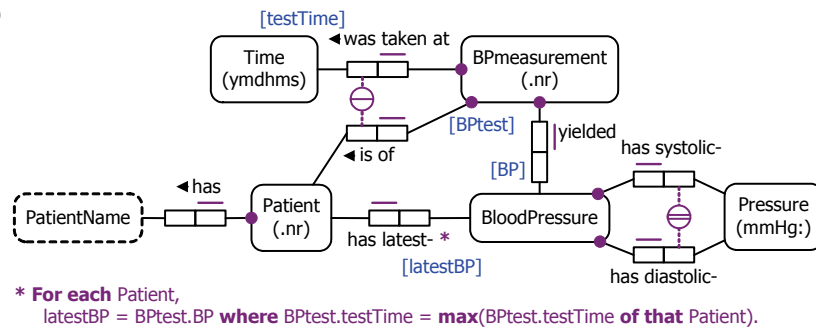


Exercise 10.3

1. (a)

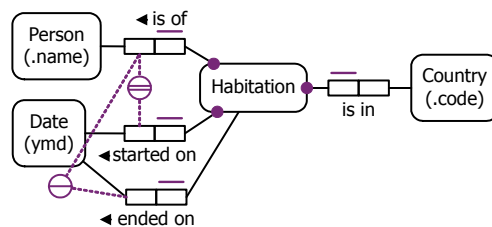


(b)



3. (a) It doesn't indicate which end date goes with which start date (this connection could be derived based on temporal ordering for a given habitation, but this is a lot of work). It doesn't capture the constraint that a person can live in at most one country for any given period. There is also an equivocation on "Habitation"—the usage in the original schema treats different occurrences of a person living in a country as the same habitation, which is not the way the term is used in the domain description.

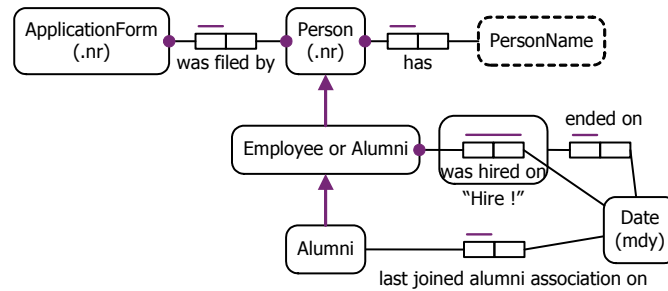
(b)



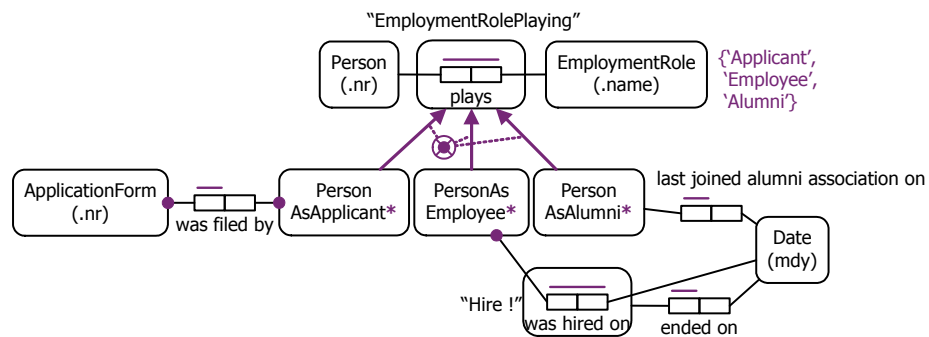
Ignore the stronger constraint that habitation periods don't overlap.
Also ignore value constraint: For each Habitation, endDate >= startDate.

5. (a) (i) If D is a subtype of B and C , it includes an instance that belongs to both B and C . But B and C are mutually exclusive. So this is impossible.
- (ii) Student is a role type, so all its instances may join or leave this type during their lifetime. If Person is a subtype of Student, when an instance leaves the Student type it also leaves the Person type. But this is impossible since Person is a rigid type (no instance can ever leave it during its lifetime).

5. (b) Using the decreasing disjunctions pattern:



Using the role-playing pattern:



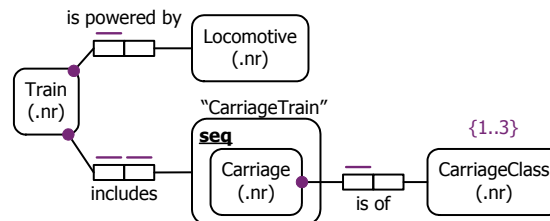
*Each PersonAsApplicant is an EmploymentRolePlaying that involves EmploymentRole 'Applicant'.

*Each PersonAsEmployee is an EmploymentRolePlaying that involves EmploymentRole 'Employee'.

*Each PersonAsAlumni is an EmploymentRolePlaying that involves EmploymentRole 'Alumni'.

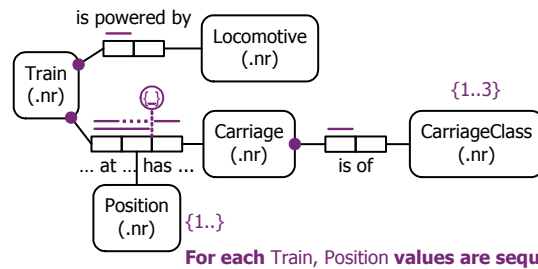
Exercise 10.4

1. (a)

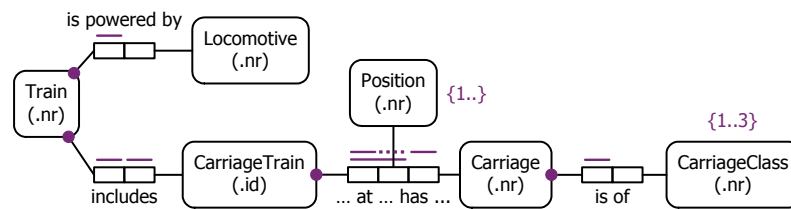


Allow CarriageTrain role to be mandatory

1. (b)

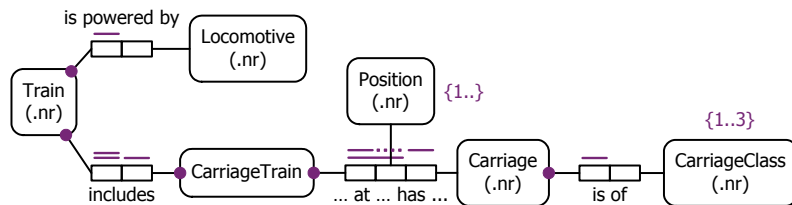


alter:

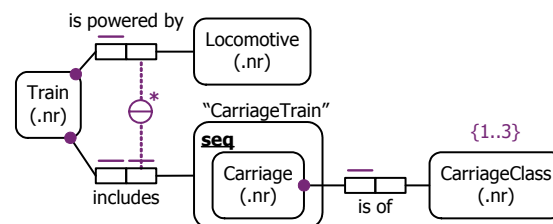


allow other CarriageTrain role to be mandatory

alter:



(c)

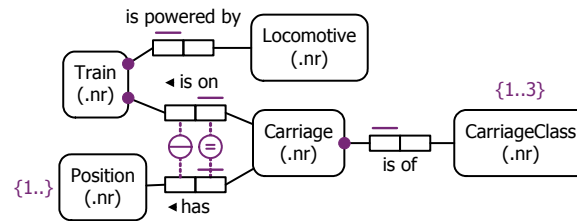


The external UC added above captures the constraint, but is not needed since it is implied by the internal uniqueness constraint on the right-hand role of the includes predicate.

(d) No for first solution. Yes for alternative solution (same explanation as for (c)).

(e) No.

1. (f)

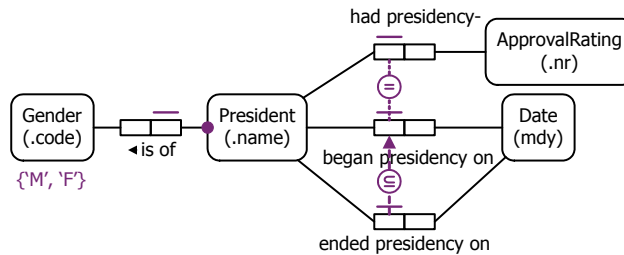


(g) No.

Exercise 10.5

1. (a) propositional
- (b) situational
- (c) propositional

3.

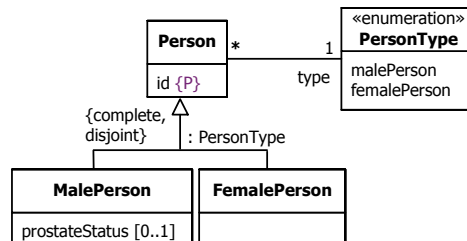


Exercise 10.6

1.	Negation Approach	Pat is male	Pat is a parent
	Negation as Failure	False	False
	Classical Negation	UNK	UNK
	Semipositive Negation	False	UNK

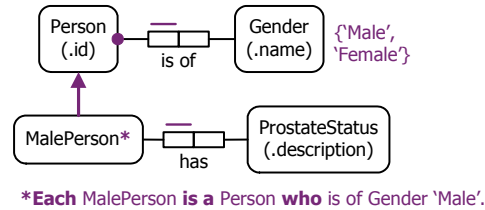
Exercise 10.7

1. (a)



- (b) We are forced to introduce the FemalePerson subclass even though it plays no specific roles. There is no formal connection between the subclasses and the enumeration values (e.g. it is possible to record prostate status for persons of type femalePerson). The personType association carries little semantics (e.g. one must inspect the enumerated values to determine anything about the classification criteria). Finally, second-order semantics are harder to implement than first-order.

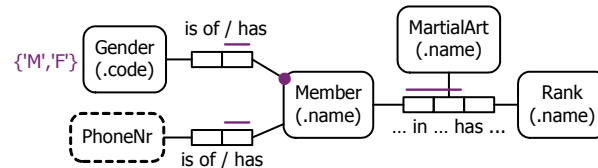
1. (c)



3. Let A denote the set of all sets that are not members of themselves. Either A is a member of A or it's not member of A . If A is a member of A , then by definition it's a set that is not a member of itself, so A is not a member of A . If A is not a member of A , then by definition A is a member of A . Hence it's both true and false that A is a member of itself, which is contradictory.

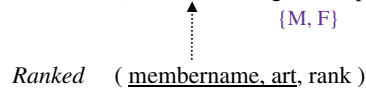
Exercise 11.2

1. (a)



- (b) The attribute “Arts and ranks” is multivalued, not atomic-valued.

(c) *Member* (membername, gender, [phone])

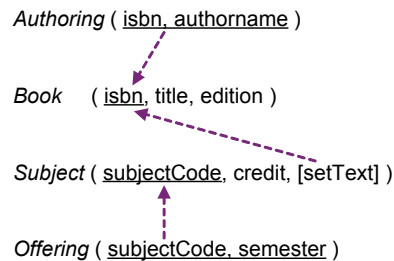


- (d) The nested relation allows all details about members to be retrieved without a table join, so is more efficient for such queries. Moreover the subset constraint is no longer needed. On the negative side, allowing nesting leads to a more complex query language, and queries to determine who has an art or rank are somewhat less efficient than for the relational solution.

As an extra exercise, modify answers (a) and (c) to unpack the semantics of ranks into a rank type (dan or kyu) and level (1..).

Exercise 11.3

1.



Information Modeling and Relational Databases: Answers (odd)–30

3. (a) The table allows redundancy. For example, with the following population the fact that dress d1 costs \$50 is duplicated:

<i>Likes</i>	(<u>woman</u> , <u>dress</u> , cost)
Eve	d1 50
Sue	d1 50

3. (b)

Likes (womanName, dressCode)

Dress (dressCode, cost, [owner])

Woman (womanName, wage)

5. (a)

Project (projectCode, mgrName, budget)

Manager (mgrName, salary, birthYear)

- (b)

Employee (empNr, deptName, salary)

Department (deptname, budget)

Note: Other facts are derived. The equality constraints above are based on the population provided. In practice, these constraints would usually be at most subset constraints.

- 7.

Branch (branchNr, branchName)

Taxpayer (taxFileNr, taxpayername, [workbranchNr]¹, [phoneNr])

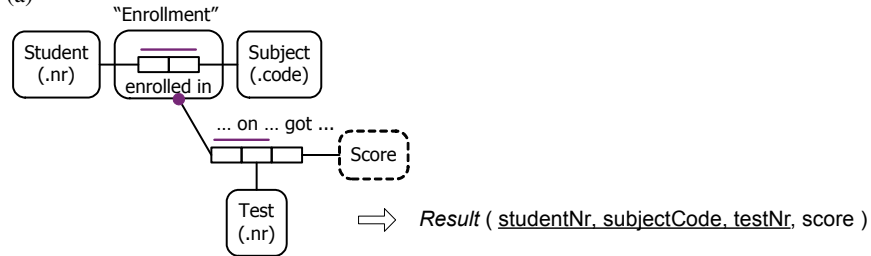
AccountUser (branchNr, accountNr, taxFileNr)

Transaction (branchNr, accountNr, tranNr, tranDate, amount, tranType, balance^{**})

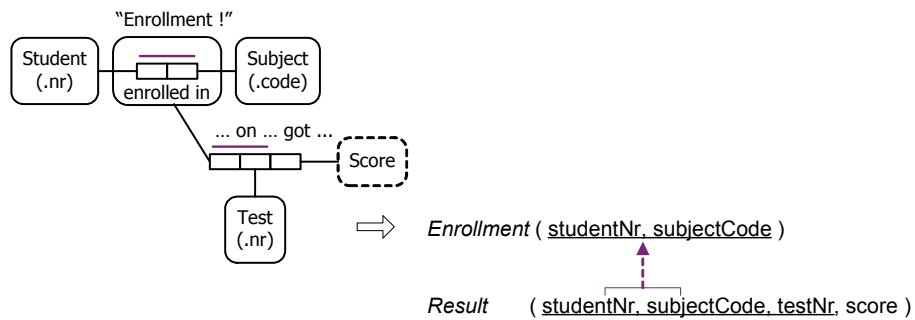
¹ null only if taxFileNr in AccountUser.taxFileNr

^{**} on create accountNr
balance := 0;
on update Transaction
if tranType = 'DEP' then balance := old balance + amount;
if tranType = 'WDL' then balance := old balance - amount

9. (a)



(b)



11. (a)

Person (formNr, age, tvHours, paperHours, [favChannel]¹, [favPaper]², [prefNews]³)

¹ exists iff tvHours > 0

² exists iff paperHours > 0

³ exists iff tvHours > 0 and paperHours > 0 and age >= 18

(b)

Person (formNr, age, tvHours, paperHours)

Reader (formNr, favPaper)

Viewer (formNr, favChannel)

MediaAdult (formNr, prefNews)

¹ exactly where tvHours > 0

² exactly where paperHours > 0

³ exactly where tvHours > 0 and paperHours > 0 and age >= 18

For most applications, schema (a) is preferable.

13.

Atmosphere (bodyName, gas)

SolarBody (bodyName, bodyKind, radius, [mass]², [orbitalPeriod]³,
[solarSeparation]⁴, [apparentMag, planetOrbited]⁵,
[nextInfConj]⁶, [nextOpposn]⁷)

¹ only where bodyKind = 'planet'
² exists iff bodyKind in {'sun','planet'}
³ exists iff bodyKind in {'planet','moon'}
⁴ exists iff bodyKind = 'planet'
⁵ exists iff bodyKind = 'moon'
⁶ exists iff solarSeparation < 1
⁷ exists iff solarSeparation > 1

* Planet (planetName, nrMoons) ::= select planetOrbited, count (*)
from SolarBody
group by planetOrbited

* 1 au = 150000000 km

15.

Country (countryName) -- includes Australia

Borders (country1, country2)¹ ¹ irreflexive

17.

Software (softwareName, producer)

SoftwareUse (softwareName, deptName)

Subject (subjectCode, deptName)

Lecturer (empNr, lecturerName, [subjectCode], deptName)

Instructs (empNr, studentSurname, studentInitials)

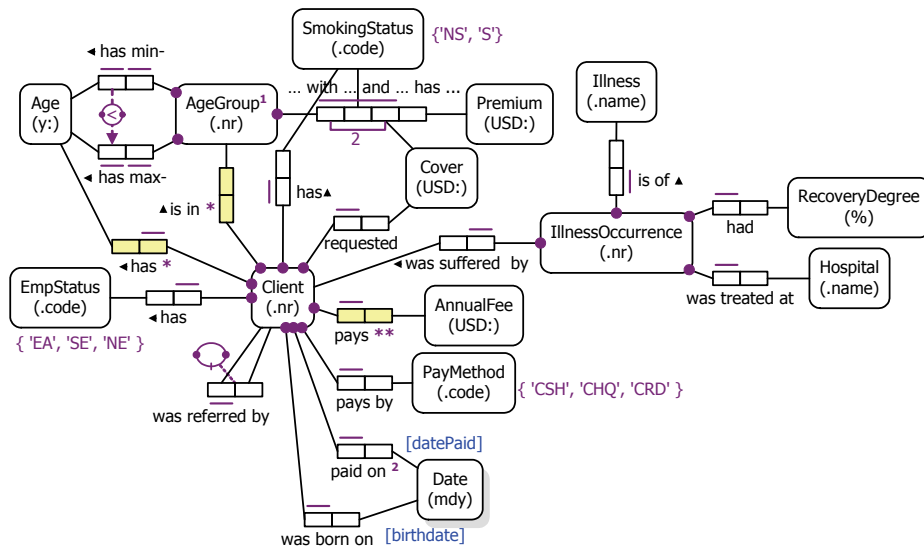
Awarded (empNr, degreeCode, uni, awardYear)

Enrollment (studentSurname, studentInitials, degreeCode, uni, completed)^{y,n}

EnrollYear (studentSurname, studentInitials, degreeCode, uni, yr, timeMode, placeMode)^{PT,'FT'} ^{'int','ext'}

Student (studentSurname, studentInitials, gender)^{M,F}

19. (a) We introduce IllnessOccurrence(.nr) to allow recording of multiple instances of the same illness by the same client. The solution makes several simplifying assumptions, and does cater for full history (e.g., typically clients are entered in the system before making payments, and clients may change their method of payment or credit cards used over time).



* For each Client, age = (today - birthdate).years.

* Client is in AgeGroup iff Client.age >= ageGroup.minAge and Client.age <= ageGroup.maxAge.

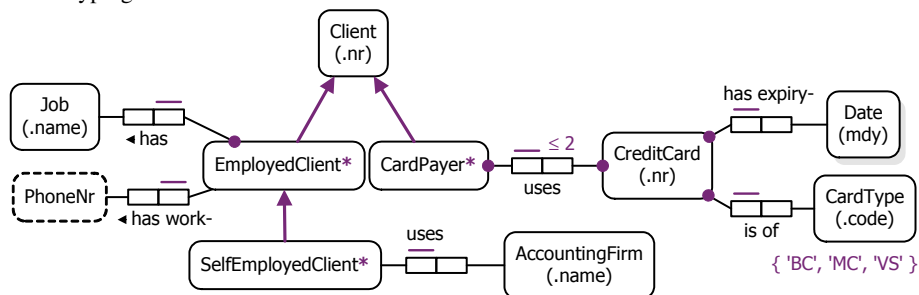
** Payment quote is derived for current date using birthdate, smoking status and proposed cover, then stored as payment if paid on that date. For simplicity, assume payment is on day of quote. Client pays AnnualFee iff

Client is in AgeGroup and has SmokingStatus and requested Cover and that AgeGroup with that SmokingStatus and that Cover has Premium and AnnualFee = 12 * that Premium.

¹ Group_{n+1}.minAge = Group_n.maxAge + 1 -- where both groups exist

² For each CardPayer, datePaid <= creditCard.expiryDate. -- this applies at time of payment

Subtyping:



*Each CardPayer is a Client who pays by PayMethod 'CRD'.

*Each EmployedClient is a Client who has EmpStatus <> 'NE'.

*Each SelfEmployedClient is a Client who has EmpStatus 'SE'.

19. (b) Even if a quoted payment were derived somehow, the actual payment would need to be stored since it might not be derivable in the future (e.g., the premium schedule might then be different, and historical records of premiums are not kept in this UoD).

(c)

AgeGroup^{1, 2} (ageGroupNr, minAge, maxAge)

Schedule (ageGroupNr, cover, smokingStatus, premium)

ClientIllness(clientNr, illness, recoveryPercent, hospitalName)

Client (clientNr, [referrer], smokingStatus, cover, payment, payMethod, datePaid³, empStatus, birthdate, [cardNr]⁴, [job]⁵, [workPhone]⁶, [accountingFirm]⁷)

CreditCard (cardNr, cardType, expiryDate)

¹ minAge < maxAge

² Group_{n+1}.minAge = Group_n.maxAge + 1

³ not exists (Client natural join CreditCard where datePaid > expiryDate)

⁴ exists iff payMethod = 'CRD'

⁵ exists iff empStatus <> 'NE'

⁶ exists only if empStatus <> 'NE'

⁷ exists only if empStatus = 'SE'

Exercise 11.4

1. (a) Politician (politicianName, isHonest)

{T, F}

(b) Politician (politicianName, [isHonest])

{T}

(c) Politician (politicianName, [isHonest])

{T, F}

(d) Student (studentNr, gender, graduated, [firstGraduationLevel]¹)

¹exists iff graduated = true

(e) Student (studentNr, gender, [graduated, firstGraduationLevel])

or

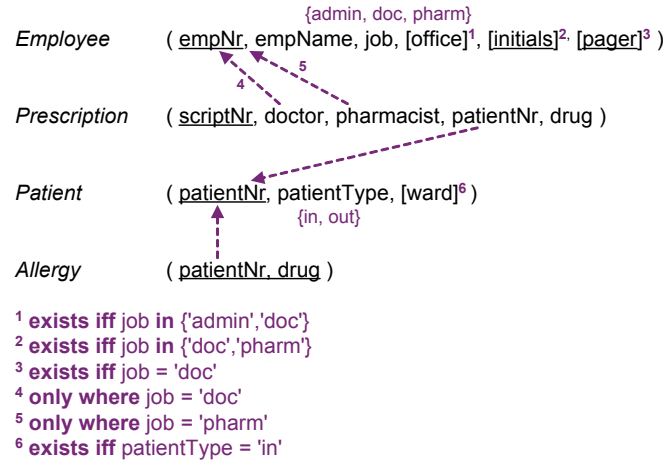
Student (studentNr, gender, [graduated, [firstGraduationLevel]¹])

¹exists iff graduated = true

(f) Student (studentNr, gender, [graduated, [firstGraduationLevel]¹])

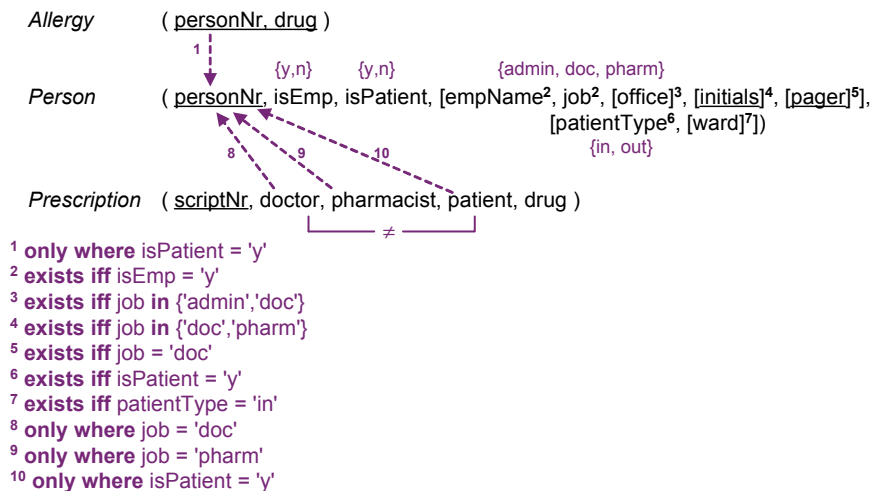
¹exists iff graduated = true

3. (a)

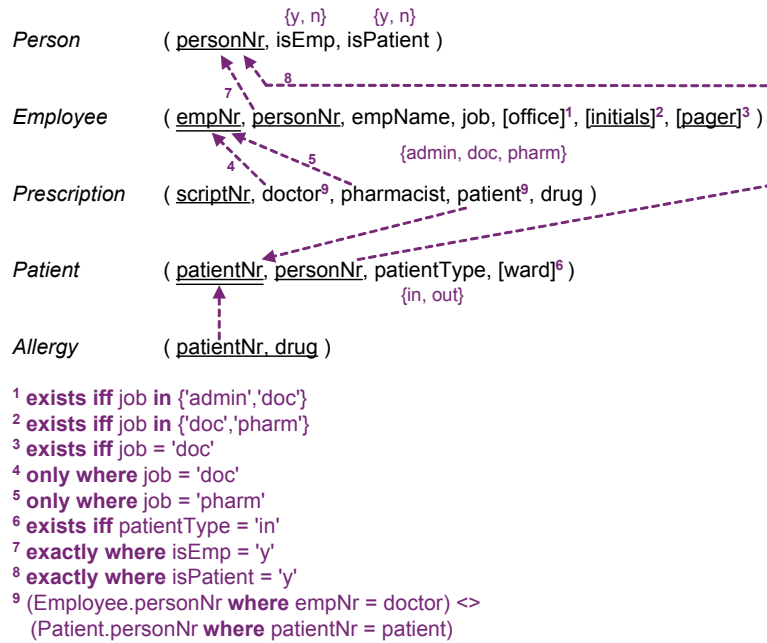


(b) Same as for (a), with documentary warning.

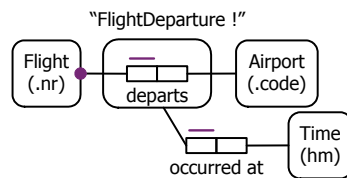
(c) This solution minimizes the number of tables by absorbing the Parent subtype. As an extra exercise, discuss the mapping if a separate Patient table is chosen.



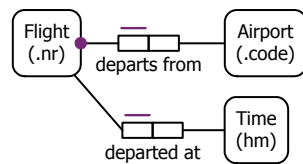
(d) The following solution assumes that personNr may still be used as a global identifier for Person, while still allowing context-dependent primary identifiers. Notice how the conceptual pair-exclusion constraint (doctors may not prescribe for themselves) has become more complicated because of the need to access the global identifiers to perform the comparison (see qualification 9, and compare this with the simple \neq check in (c)).



5. (a) This assumes departure time might not be known. If it must be known, add a mandatory constraint and remove the independence setting for FlightDeparture.



(b)



- (c) Flight (FlightNr, origin, [departureTime])

Exercise 12.1

1. (a)	<i>Uses</i> (<u>student</u> , computer)	<i>Owns</i> (<u>student</u> , computer)
	Ann P2	Fred P2
	Fred P2	Sue P1
	Fred Mac	Tom Mac
	Sue P3	

(b)	$(\text{Uses} \cup \text{Owns}) [\text{student}]$	\rightarrow	<i>student</i>
	<i>alter:</i> $\text{Uses} [\text{student}] \cup \text{Owns} [\text{student}]$		Ann Fred Sue Tom

Note: To save space, later results are shown on one line with the column name omitted

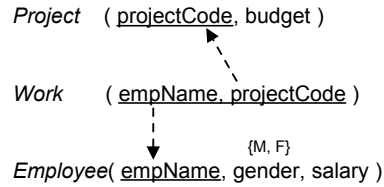
(c)	$(\text{Uses} - \text{Owns}) [\text{student}]$	$\rightarrow \{ \text{Ann, Fred, Sue} \}$
(d)	$\text{Uses} [\text{student}] - \text{Owns} [\text{student}]$	$\rightarrow \{ \text{Ann} \}$
(e)	$\text{Owns} [\text{student}] - \text{Uses} [\text{student}]$	$\rightarrow \{ \text{Tom} \}$
(f)	$(\text{Owns} - \text{Uses}) [\text{student}]$	$\rightarrow \{ \text{Sue, Tom} \}$
(g)	$(\text{Uses} \cap \text{Owns}) [\text{student}]$	$\rightarrow \{ \text{Fred} \}$
(h)	$\text{Uses} [\text{student}] \cap \text{Owns} [\text{student}]$	$\rightarrow \{ \text{Fred, Sue} \}$
(i)	$\text{Uses} [\text{computer}] - \text{Owns} [\text{computer}]$	$\rightarrow \{ \text{P3} \}$
(j)	$\text{Owns} [\text{computer}] - \text{Uses} [\text{computer}]$	$\rightarrow \{ \text{P1} \}$

3. (a) Subject **where** subjCode = 'CS113'
 (b) (Student **where** gender = 'M' **and** birthYr > 1960)
 /studentNr, studentName, degree/
 (c) (Student \bowtie Result) **where** studentName = 'Brown T' /subjCode/
 alter: ((Student **where** studentName = 'Brown T') /studentNr/
 \bowtie Result) /subjCode/
 alter: ((Student **where** studentName = 'Brown T') \bowtie Result) /subjCode/

Note: From now on we normally show only one solution, favoring joins first if convenient, and omitting optional intermediate projections.

- (d) (Student \bowtie Result) **where** rating = 7
 /studentNr, studentName/
 (e) (Student \bowtie Subject \bowtie Result)
 where title = 'Logic' **and** rating = 5
 /studentNr, studentName/
 (f) ((Result /studentNr, subjCode/ \div Subject /subjCode/) \bowtie Student)
 /studentNr, degree/
 (g) (Student **where** degree = 'BSc') /studentNr, studentName, gender/
 \cup
 ((Result **where** subjCode = 'PD102' **and** rating = 7) \bowtie Student)
 /studentNr, studentName, gender/
 (h) (Student \bowtie Subject \bowtie Result)
 where gender = 'M' **and** birthYr < 1970
 and rating >= 5 **and** title = 'Databases'
 /studentNr, studentName, birthYr/

5. (a)



(b) (Work ⋈ Employee)
where gender = 'F' **and** (salary > 25000 **or** projectCode = '5GIS')
 [empName, salary]

(c) (Employee ⋈ Work) [empName, gender, projectCode]
 ÷
 Project **where** budget >= 100000 [projectCode]

7. (a) $A \cap B = A - (A - B) \quad \{ \text{ or } B - (B - A) \}$

(b) $A \bowtie B = (A \times B) \text{ where } A.y = B.y \text{ [} A.x, A.y, B.z \text{]}$

(c) $A - B = A[x] - ((A[x] \times B) - A)[x]$

9. (a) (Invoice ⋈ LineItem ⋈ Item)

where unitPrice < listPrice [customerNr/

–

(Invoice ⋈ LineItem ⋈ Item)

where category = 'WP' [customerNr/

Note: It is wrong to simply add: “**and** category <> 'WP'”.

(b) ((Invoice ⋈ LineItem) [customerNr, itemCode/

÷

(Item **where** category = 'SS') [itemCode/

⋈

Customer) [customerName/

(c) (Invoice ⋈ LineItem) [customerNr, itemCode/

÷

(Item **where** category = 'WP') [itemCode/

–

(Invoice ⋈ LineItem ⋈ Item)

where category = 'DB' [customerNr/

(d) (LineItem ⋈ Item) **where** category = 'SS' [invoiceNr/

∩

(LineItem ⋈ Item) **where** category = 'WP' [invoiceNr/

⋈ Invoice [customerNr/

Exercise 12.4

1. (a) Illegal SQL-89, SQL-92, SQL:1999, and SQL:2003 (but legal in some commercial dialects)
- (b) Legal in all
- (c) Legal as a delimited identifier in SQL-92, SQL:1999 and SQL:2003
- (d) Illegal in all (starting digit)
- (e) Illegal in all (embedded space)
- (f) Legal as a delimited identifier in SQL-92, SQL:1999, and SQL:2003
- (g) Legal in all
- (h) Illegal in all (but legal in some commercial dialects, which allow “\$”)
- (i) Illegal in all (parentheses)
- (j) Illegal in all (reserved word)
- (k) Illegal in SQL-89 (too long) but legal in SQL-92, SQL:1999, and SQL:2003
- (l) Legal in SQL:1999 and SQL:2003 (where “count” is no longer a reserved word)

Exercise 12.5

1. (a) **select** kind **from** Statement
- (b) **select** * **from** Statement
- (c) **select** kind, composition **from** Statement
- (d) **select** kind, extra **from** Statement
- (e) **select distinct** composition **from** Statement
- (f) **select** kind **from** Statement
where composition = 'structured'
- (g) **select** kind **from** Statement
where composition = 'simple'
and extra = 'Y'
- (h) **select** kind, composition
from Statement
where extra = 'Y'
- (i) **select** kind **from** Statement
where extra = 'N'
order by kind
- (j) **select** kind, composition
from Statement
order by composition
- (k) **select** kind, composition **from** Statement
order by composition **desc**, kind **asc** -- **asc** may be omitted since it is implied
- (l) **select** * **from** Statement
order by extra **desc**, composition **asc**, kind **desc**
- (m) **select** kind **from** Statement
where composition <> 'structured' **and** extra = 'N'
order by kind
- (n) **select** kind **from** Statement
where composition = 'structured' **and** extra = 'Y'
or composition = 'simple'
order by kind **desc**
- (o) **select** kind, composition **from** Statement
where kind <> 'exit'
and (composition = 'structured' **and** extra = 'Y'
or composition = 'simple')
order by composition **desc**, kind **desc** -- the parentheses in this query are needed

Exercise 12.6

1. (a) **select** studentNr, studentName, degree, birthYr **from** Student
order by degree, birthYr
- (b) **select** studentNr, subjCode
from Result **natural join** Student
where studentName = 'Smith J'

Note: If the newer join syntax introduced in SQL-92 is not supported, use older syntax, e.g.,

```

select Result.studentNr, subjCode
from Result join Student
      on Result.studentNr = Student.studentNr
where studentName = 'Smith J'
or
select Result.studentNr, subjCode
from Result, Student
where Result.studentNr = Student.studentNr and studentName = 'Smith J'
    
```

Here, “Result.studentNr” in the select-list may be replaced by “Student.studentNr” but not just “studentNr”. Later solutions assume the SQL-92 join syntax is supported.

- (c) **select** studentNr, studentName, gender
from Student **natural join** Result
where subjCode = 'CS113'
- (d) **select** title **from** Subject **natural join** Result
where studentNr = 863 -- assuming studentNr has a numeric data type
- (e) **select** studentNr, studentName, degree
from Student **natural join** Subject **natural join** Result
where gender = 'M' **and** rating = 5 **and** title = 'Logic'
order by studentName
- (f) **select distinct** subjCode, credit
from Student **natural join** Subject **natural join** Result
where gender = 'M' **and** degree = 'BSc' **and** rating = 7
order by credit **desc**, subjCode
- (g) **select** subjCode, title, rating
from Subject **natural left join** Result

Exercise 12.7

1. (a) **select** title **from** CompLanguage
where releaseYr **in** (1959,1975,1979)
- (b) **select** title **from** CompLanguage
where releaseYr **between** 1959 **and** 1979
-- Logo and SQL absent because of nulls
- (c) **select** title **from** CompLanguage
where releaseYr **is null**
- (d) **select** * **from** CompLanguage
where title **like** 'Goodo%'
- (e) **select** title **from** CompLanguage
where title **like** '%OL'
- (f) **select** * **from** CompLanguage
where title **like** '%Pascal%'
- (g) **select** title **from** CompLanguage
where title **like** '_____' { 5 _ }
- (h) **select** title **from** CompLanguage
where title **like** '_o%'

1. (i) **select title from** CompLanguage
where title like '%O%' **or title like** '%o%'
- (j) **select * from** CompLanguage
where title not like 'P%'
and releaseYr between 1960 **and** 1978
- (k) **select title from** CompLanguage
where title like '____a' **or title like** '____a_' -- 5 then 4,1 underscore
or title like '____OL' -- 3 underscores

Exercise 12.8

1. (a) **select person from** Eater **where** gender = 'M' **or** weight > 60
- (b) **select person from** Eater **where** gender = 'M'
union
select person from Eats **where** foodName = 'peas'
aliter: **select person from** Eater
where gender = 'M' **or**
person **in** (**select person from** Eats
where foodName = 'peas')
- aliter:* **select distinct** person
from Eater **natural join** Eats
where gender = 'M' **or** foodName = 'peas'
- (c) **select person from** Eater **where** gender = 'F' **and** weight > 60
- (d) **select person from** Eater **where** gender = 'F'
intersect
select person from Eats **where** foodName = 'potato'
aliter: **select person from** Eater
where gender = 'F' **and** person **in**
(**select person from** Eats
where foodName = 'potato')
- (e) **select person from** Eater **where** weight > 60
except
select person from Eats **where** foodName = 'beef'
aliter: **select person from** Eater
where weight > 60 **and** person **not in**
(**select person from** Eats **where** foodName = 'beef')
- (f) **select** A.person, B.person
from Eater **as** A, Eater **as** B
where A.weight = B.weight **and** A.gender < B.gender
Note: This is the neatest way to avoid showing each pair in two different orders. Here the female of each pair appears on the left. A correct but longer solution is to order members of each pair alphabetically. For example:

select A.person, B.person **from** Eater **as** A, Eater **as** B

where A.weight = B.weight
and A.gender <> B.gender
and A.person < B.person

- (g) **select distinct person from Eats**
where foodName **in**
 (select foodName **from** Food
 where foodClass = 'vegetable')
- (h) **select distinct person, weight**
from Eater **natural join** Eats **natural join** Food
where gender = 'M' **and** foodClass = 'meat'
aliter: **select person, weight from Eater**
where gender = 'M' **and** person **in**
 (select person **from** Eats
 where foodName **in**
 (select foodName **from** Food
 where foodClass = 'meat'))
- (i) **select distinct person from Eats**
where foodName **not in** (select foodName **from** Food
 where foodClass = 'meat')
- (j) **select person from Eater**
where person **not in**
 (select person **from** Eats **where** foodName **in**
 (select foodName **from** Food **where** foodClass = 'meat'))
- (k) **select** 'vegetarian: ' **as** eaterType, person, weight, gender **from** Eater
where person **not in**
 (select person **from** Eats **where** foodName **in**
 (select foodName **from** Food **where** foodClass = 'meat'))
union
select 'meateater: ' **as** eaterType, person, weight, gender **from** Eater
where person **in**
 (select person **from** Eats **where** foodName **in**
 (select foodName **from** Food **where** foodClass = 'meat'))
order by eaterType **desc**, gender, weight, person

Note: From SQL-92 onwards, the list “person, weight, gender” may be replaced by “Eater.*”.

Exercise 12.9

1. (a) **select count(*) from Eater**
where gender = 'M' **and** weight > 100
- (b) **select count(distinct weight) from Eater**
- (c) **select sum(weight) from Eater**
where gender = 'M'
- (d) **select avg(weight) from Eater where gender = 'F'**

- (e) **select max(weight) from Eater**
where person in
 (select person from Eats
 where foodName = 'beef')
- (f) **select person from Eater** *aliter:* **select person from Eater**
where gender = 'F' and weight < **where gender = 'F' and weight < all**
 (select min(weight) from Eater **(select weight from Eater**
 where gender = 'M') **where gender = 'M')**
- (g) **select person, weight from Eater**
where gender = 'F' and weight >=
 (select min(weight) from Eater
 where gender = 'M')
- aliter:* **select person, weight from Eater**
where gender = 'F' and weight >= some
 (select weight from Eater where gender = 'M')

3. *Note:* In this database the nationalities of authors from America, Australia and New Zealand are stored as 'American', 'Aussie', and 'Kiwi', respectively.

- (a) **select authorName from Author**
where gender = 'M' and nationality = 'Aussie'
- (b) **select title from Book**
where yrPublished between 1984 and 1986
order by title
- (c) **select * from Book**
where title like '%SQL%'
order by yrPublished desc
- (d) **select authorName from WrittenBy**
where isbn in
 (select isbn from Book
 where title = 'Databases' and yrPublished = 1980)

aliter: **select authorName from Book natural join WrittenBy**
where title = 'Databases' and yrPublished = 1980

The previous alternative solutions use the subquery approach and the join approach. The remaining solutions use the subquery approach. As an extra exercise, formulate the equivalent solutions using joins.

- (e) **select isbn, title from Book**
where isbn in
 (select isbn from WrittenBy
 where authorName in
 (select authorName from Author where nationality = 'Aussie'))
- (f) **select authorName, nationality from Author**
where authorName in

```
(select authorName from WrittenBy
where isbn in
  (select isbn from Book
   where title = 'Informatics' and publisher = 'Hall' and yrPublished = 1986))
```

- (g) **select** 'Australian male: ' **as** personType, authorName
from Author
where gender = 'M' **and** nationality = 'Aussie'
union
select 'American female: ' **as** personType, authorName
from Author
where gender = 'F' **and** nationality = 'American'
order by personType **desc**

Exercise 12.10

1. (a) **select** gender, **count**(*) **from** Author
group by gender
- (b) **select** gender, nationality, **count**(*) **as** nrAuthors
from Author
group by gender, nationality
order by nrAuthors **desc**
- (c) **select** publisher, **sum**(nrCopies) **from** Book
where yrPublished > 1980
group by publisher
- (d) **select** publisher, **min**(yrPublished) **from** Book
group by publisher
having avg(nrCopies) > 2
- (e) **select** isbn, **count**(*) **from** WrittenBy
where isbn in
 (select isbn **from** Book **where** yrPublished < 1986)
group by isbn
having count(*) > 1
- (f) **select** nationality, **count**(*) **from** Author
where gender = 'M' **and** authorName in
 (select authorName **from** WrittenBy
 where isbn in
 (select isbn **from** Book
 where publisher in ('Hall','Possum'))))
group by nationality
having count(*) >=
 (select nrCopies **from** Book
 where isbn = '444')

Exercise 12.11

1. (a) **select** pupilNr, surname, firstname
from Pupil
where iq =
 (select max(iq) **from** Pupil)
- (b) **select** pupilNr, surname, firstname
from Pupil **as** X
where iq =
 (select max(iq) **from** Pupil
 where gender = X.gender)
- (c) **select** M.pupilNr, M.surname, M.firstname, F.pupilNr, F.surname, F.firstname
from Pupil **as** M, Pupil **as** F
where M.gender = 'M' **and** F.gender = 'F'
 and M.iq = F.iq
3. (a) **select** studentNr, studentName
from Student
where studentNr **not in**
 (select studentNr **from** Result
 where rating = 7)
- (b) **select** studentNr, studentName
from Student **natural join** Result
group by studentNr, studentName
having max(rating) < 7

Note: The following query is *wrong*, since it also returns those get both a 7 and a non-7 rating.

```
select studentNr, studentName
from Student natural join Result
where rating <> 7
```

Exercise 13.3

1. **create table** Movie (
 movieCode **char**(4) **not null** **primary key**,
 title **varchar**(20) **not null**,
 category **char**(2) **not null** **check** (category in ('G', 'PG', 'M', 'R')));

create table VideoTape (
 movieCode **char**(4) **not null** **references** Movie,
 copyNr **dec**(2) **not null**,
 purchaseDate **date** **not null**,
 status **char**(2) **not null** **check** (status in ('Good', 'OK', 'Poor', 'X')),
 comment **varchar**(20),
 writtenOff **char** **check** (writtenOff in ('Y', 'N'),
 primary key (movieCode, copyNr),
 check (status <> 'X' **and** comment **is null** **and** writtenOff **is null**) **or**
 (status = 'X' **and** comment **is not null** **and** writtenOff **is not null**));

```

create table Customer (
  cardNr          smallint      not null primary key,
  customerName    varchar(20)   not null,
  address         varchar(30)   not null,
                unique (customerName, address) );

create table Loan (
  loanNr          smallint      not null primary key,
  issueDate       date         not null,
  cardNr          smallint      not null references Customer );

create table LoanItem (
  loanNr          smallint      not null references Loan,
  movieCode       char(4)      not null,
  copyNr          smallint      not null,
  cost            dec(5,2)      not null,
  returnDate      date,
                primary key (loanNr, movieCode, copyNr),
                foreign key (movieCode, copyNr) references VideoTape );

create assertion each_loan_has_a_loan_item
check( not exists
      (select * from Loan
       where loanNr not in
         (select loanNr from LoanItem)) )

```

3. (a) **create table** Technician (
- ```

 empNr int not null primary key,
 firstName varchar(20) not null,
 lastName varchar(30) not null,
 supervisor int
 foreign key references Technician(empNr))

create table Assembly (
 assemblyCode char(10) not null primary key,
 complexityNr smallint not null
 check (complexityNr between 0 and 8),
 partOf char(10)
 foreign key references Assembly(assemblyCode),
 isTopLevel char not null
 check (isTopLevel in ('Y', 'N')), -- see answer to 3(c)
 hasReport char not null
 check (hasreport in ('Y', 'N')))

create table Competence (
 empNr int not null
 foreign key references Technician(empNr),
 assembly char(10) not null
 foreign key references Assembly(assemblyCode),
 certificationRef varchar(20) unique,
 primary key(empNr, assembly))

```

Other possibilities exist. For example, constraints can be named for convenience. Also, variations between SQL implementations may introduce other features. For example, it might be wise to make “assembly” a delimited identifier in SQL Server to avoid confusion with the .NET integration features.

- (b) The foreign key references in the “create table” statements impose some ordering, so the Competence table must be created after the Technician and the Assembly tables. For similar reasons, the Technician and Assembly tables cannot be dropped while the foreign key references exist in the Competence table. One way around the ordering constraints on table creation is to omit the foreign key references in the “create table” statement and add them later using “alter table...”.
  - (c) The self-references in the Technician and Assembly tables give clues to the existence of hierarchies. In the case of a technician at the top of a hierarchy (there could be more than one hierarchy) the “supervisor” reference will be null. However, this value could also be null simply because we don’t currently know who supervises a particular technician. The Assembly hierarchy provides an explicit “isTopLevel” flag to indicate a node at the top of a hierarchy. It would be wise to include an additional check to ensure that “partOf” is null if “isTopLevel” is ‘Y’.
5. (a) This is not a correct statement. Although ordinary views can be visualized as tables (and queried in much the same way), no table actually exists. A queries on a view is integrated with the view’s stored query definition and the combination is applied to base tables. Unlike ordinary views, materialized views are actually stored.
- (b) This is not a correct statement. A view definition may reference another view (sometimes referred to as a “nested” view. Some SQL dialects also allow recursive views.
  - (d) This is a correct statement. However the amount of data storage required to hold a view definition is miniscule compared with the typical storage required for data in the base tables.
  - (e) This is not a correct statement. Views are not materialized unless they are explicitly specified as such.
  - (f) This is a correct statement. Database systems limit the updating of views to updates that are “sensible” in the context of the view definition. It may be impossible for an automated system to work out the intent of an update when the view has been constructed from, say, a join between base tables.

#### Exercise 13.4

1. We need at least *two* triggers to reproduce the functionality of the “references” clause: one at the *referencing* end and one at the *referenced* end. The examples here are for SQL Server. There may be minor differences for other dialects of SQL.

```
-- check that a Competence refers to a valid Technician
create trigger CompetenceHasTechnician
on Competence
after insert, update
as
if exists
```

```
(select *
from Inserted as i left outer join Technician as t
on i.empNr = t.empNr
where t.empNr is null)
begin
 raiserror('Competence must refer to valid technician', 16, 1)
 rollback transaction
end
```

-- if a Competence refers to a Technician we do NOT want  
-- to delete the Technician or change the key column

```
create trigger TechnicianHasCompetence
on Technician
after delete, update
as
if exists
```

```
(select *
from Deleted as d join Competence as c
on d.empNr = c.empNr
left outer join Inserted as i
on d.empNr = i.empNr
where i.empNr is null)
```

```
--
begin
 raiserror('Technician is referenced: Modification failed', 16, 1)
 rollback transaction
end
```

3. **create trigger** ComplexityAdjustment

```
on Assembly
after update
as
if exists
 (select *
 from Inserted as i join Deleted as d
 on i.assemblyCode = d.assemblyCode
 where abs(i.complexityNr - d.complexityNr) > 1)
 -- abs(expr) is a function that returns the absolute value of an expression
begin
 raiserror('complexityNr can only change by 1: Update failed', 16, 1)
 rollback transaction
end
```

Some optimizations are possible – for example, checking to see if any rows were actually affected by the attempted trigger action before incurring the expense of running the trigger body – but we ignore such possibilities here.



### Exercise 13.5

The solutions here assume SQL Server. The syntax for other SQL implementations may vary.

1. A tricky question. The nature of the reference from supervisor to empNr means that an empNr for a supervising technician must already be stored before it can be referenced as a supervisor. The only way that a loop could be created with an insert is if a reference to the empNr of the new row already existed – which is clearly impossible since the row hasn't yet been inserted! However, we certainly should implement a check on acyclic relationships such as this because an *update* to an existing row in the table *could* create a loop.

The following stored procedure shows an example of this kind of check. To make sure that the check is applied on every update we could call the stored procedure from an “after update” trigger. The stored procedure has two parameters, the second of which has a default value of null. The procedure is called with just one parameter (the empNr to be checked); the second is used during recursion. The procedure returns 99 if a loop is detected and 0 otherwise.

```
create proc sp_TechnicianCheck
@baseEmp int,
@thisEmp int = null
as
if @baseEmp = @thisEmp return 99 -- referential loop detected
declare @newSupr int, @supervisee int, @rtn int
if @thisEmp is null set @supervisee = @baseEmp
 else set @supervisee = @thisEmp
select @newSupr = supervisor from Technician where empNr = @supervisee
if @newSupr is null return 0 -- top of hierarchy
exec @rtn = sp_TechnicianCheck @baseEmp, @newSupr -- otherwise go up one level
return @rtn
```

3. **create function** AmountWithcurrency  
 (@code **char**(3), @amount **decimal**(9,2))  
**returns varchar**(11)  
**as**  
**begin**  
     **declare** @symbol **char**, @rate **float**  
     **select** @symbol = symbol **from** Currency **where** code = @code  
     **select** @rate = rate **from** Currency **where** code = @code  
     **return** @symbol + **cast**((@amount \* @rate) **as varchar**(10))  
     -- the result is up to 11 characters long in this case  
     -- that's the symbol (1) plus the value (up to 9 digits) plus the decimal point (1)  
**end**

### Exercise 13.9

1. The syntax to produce XML from database tables varies between implementations. Here are examples for SQL Server and DB2. Both of these produce the same structure: a root node called “myTechnicians” containing a series of elements called “technician”, each of which contains elements for empNr, lastName and supervisor (the supervisor element will be missing if the corresponding relational value is null).  
 -- SQL Server

```
select empNr, lastName, supervisor
from technician
for xml auto, elements,
root ('myTechnicians')
```

-- DB2

```
select xmlelement (name "myTechnicians",
 xmllagg(xmlelement(name "technician",
 xmlforest
 (t.empNr as "empNr",
 t.lastName as "lastName",
 t.supervisor as "supervisor")))))
from technician as t
```

3. Relational tables can be generated from XML, but, again the syntax varies from one implementation to another. The following examples for SQL Server and DB2 assume that we already have an XML document called “myTechnicians”.

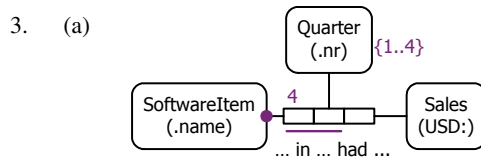
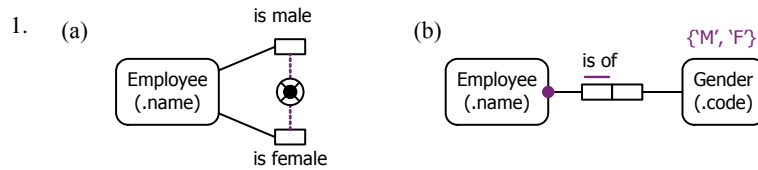
-- SQL Server

```
select T.c.value('empNr[1]', 'int') as empNr,
 T.c.value('lastName[1]', 'varchar(30)') as lastName,
 T.c.value('supervisor[1]', 'int') as supervisor
from @myTechnicians.nodes('/myTechnicians/technician') T(c)
```

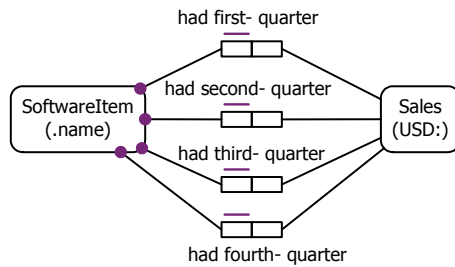
-- DB2

```
xmltable ('//technician'
passing myTechnicians
columns
 "empNr" int path 'empNr',
 "lastName" varchar(30) path 'lastName',
 "supervisor" int path 'supervisor')
```

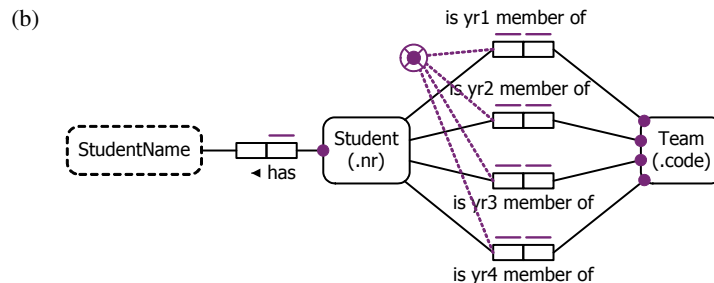
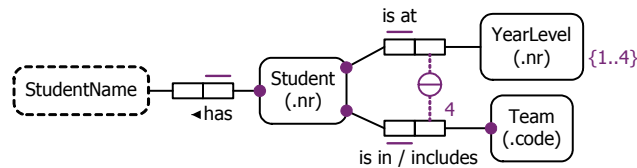
**Exercise 14.2**



(b) Notice the hyphen binding (the hyphen word and any following words are bound to the next object type term). For example, the constraints on the first fact type verbalize as: **Each** SoftwareItem had **exactly one** first quarter Sales.



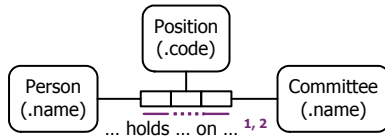
5. (a) This solution assumes all students are team members and all teams are full.



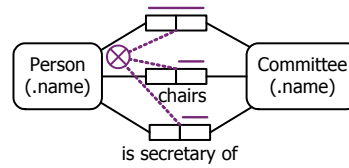
As a further exercise, discuss what happens if it is optional for a student to belong to a team.

7.

(1) {chr', 'sec', 'ord'}



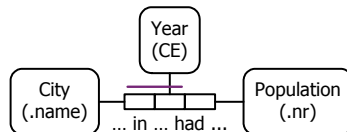
(2) is an ordinary member of



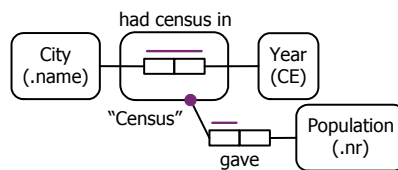
- <sup>1</sup> For each Committee,  
at most one Person holds Position 'chr' on that Committee.  
<sup>2</sup> For each Committee,  
at most one Person holds Position 'sec' on that Committee.

### Exercise 14.3

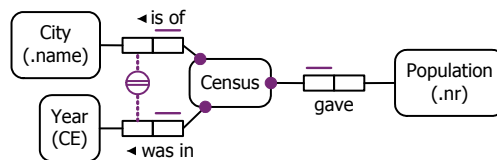
1. (a)



(b)

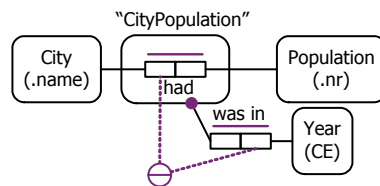


(c)



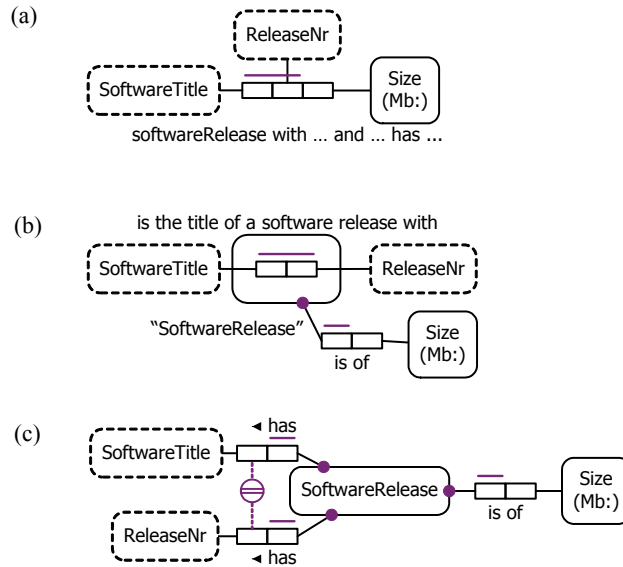
(d) I prefer (a). How about you?

(e)

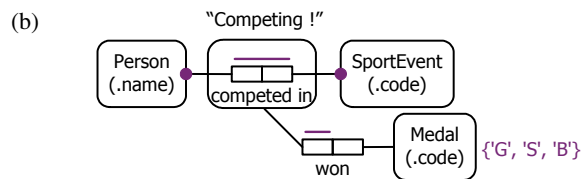


This is worse.

3. I prefer the co-referenced version. How about you? Also the software size should more realistically have been measured in megabytes (MB) not megabits (Mb).



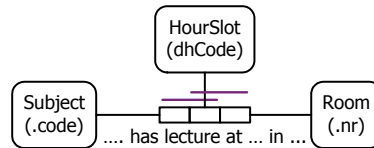
5. (a) *Competing* ( personName, sportEvent )
- Win* ( personName, sportEvent, medal ) {G, S, B}



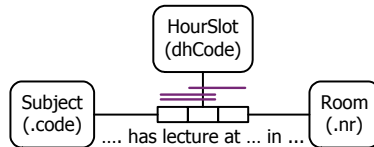
- (c) *Competing* ( personName, sportEvent, [medalWon] ) {G, S, B}

- (d) I prefer the nested conceptual schema. How about you?

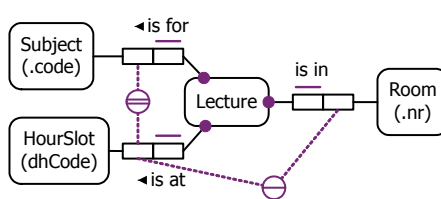
7. (a)



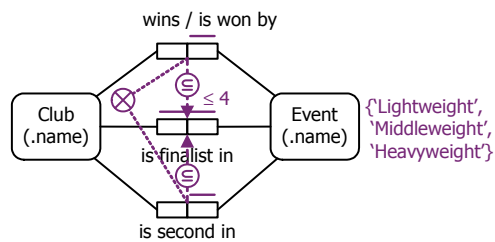
*Note:* For Rmapping, one UC may be annotated as preferred (double-line) to show choice of primary key, e.g.



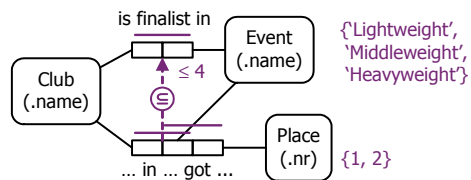
(b)



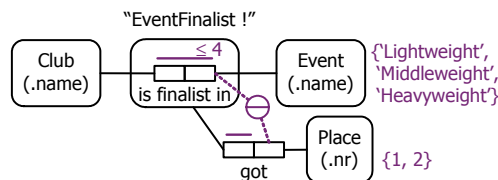
9. (a)



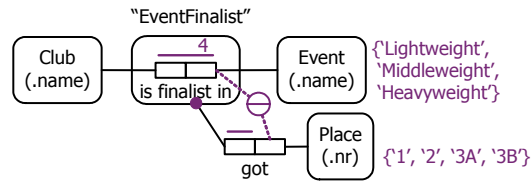
(b)



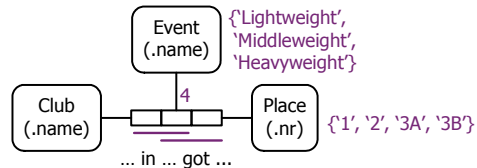
(c)



9. (d) This solution assumes that if any results are recorded at all, then all the final results must be recorded (i.e. record all results of the completed competition or record no results).



(e)



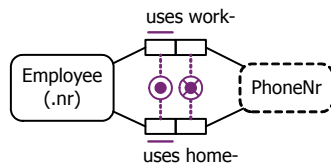
- (f) I prefer solution (e) to (d). How about you?

#### Exercise 14.4

1. (a)

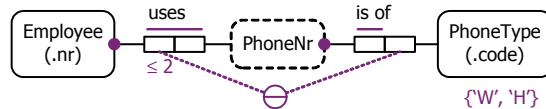


(b)

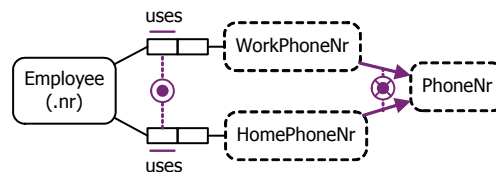


*Note:* If we remove the mandatory component of the xor constraint we need to restore the fact type Phone is of PhoneType { 'W', 'H' } to preserve phone type information

(c)

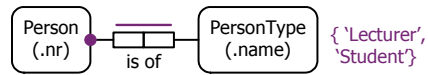


(d)

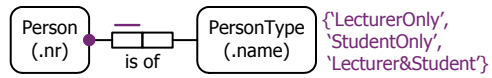


Here the subtypes are asserted (to instead derive them, add the PhoneType fact type).

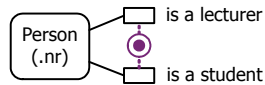
3. (a)



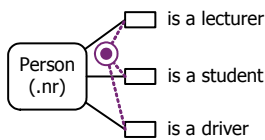
(b)



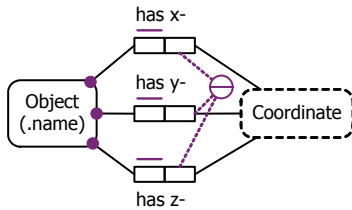
(c)



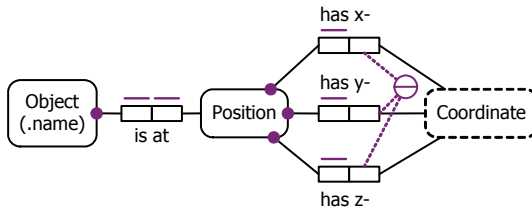
(d)



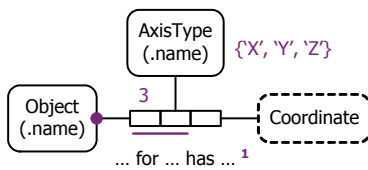
5. (a)



(b)



(c)



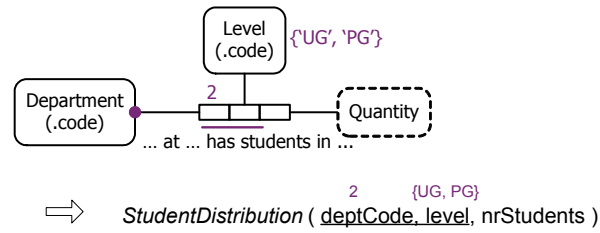
<sup>1</sup> At most one Object has the same Coordinate set.

Note: The textual constraint is somewhat expensive to enforce.

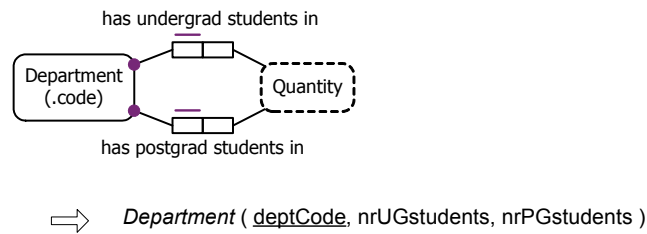


**Exercise 14.5**

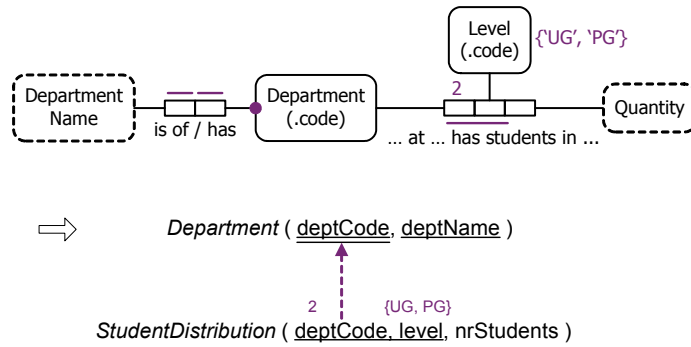
1. (a)



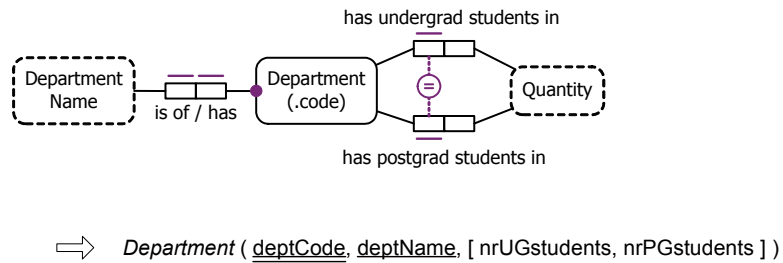
(b)



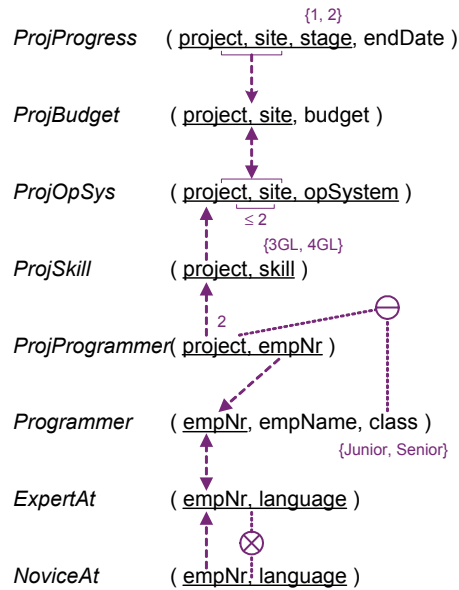
1. (c)



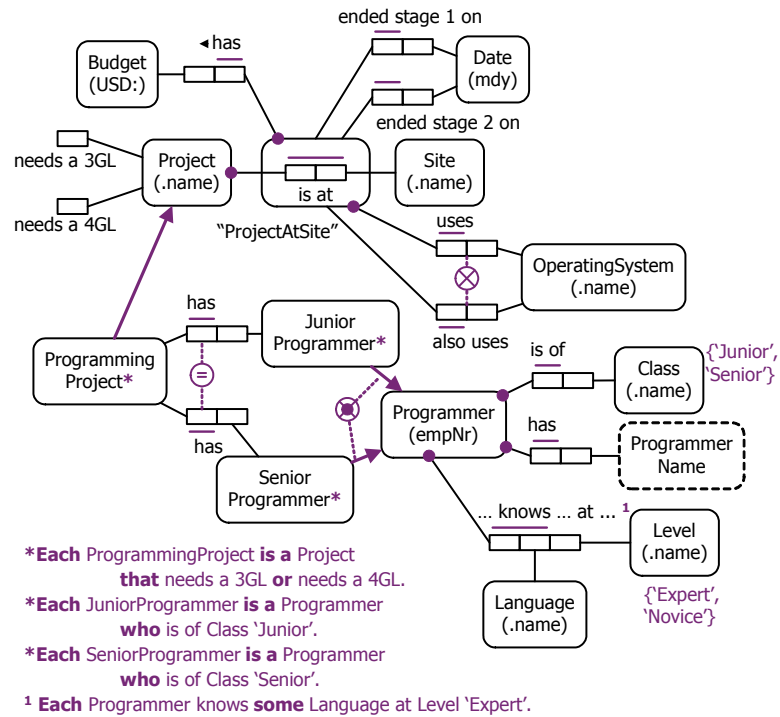
(d)



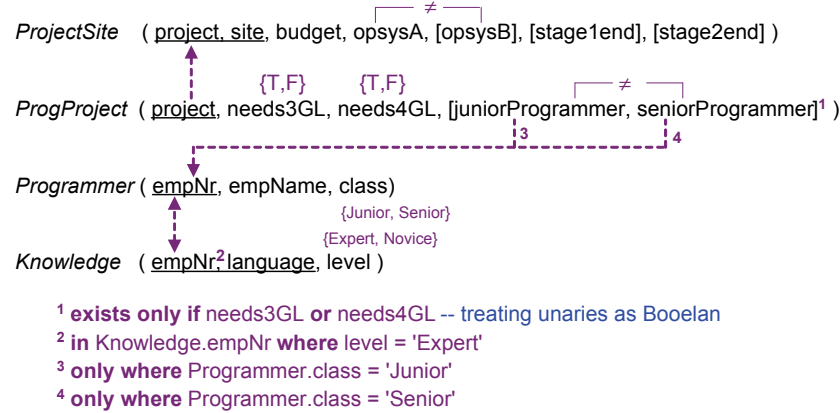
3. (a)



3. (b) For the *uses* and *employs* predicates, the schema has been strengthened to make the first of these mandatory (if this is not done, a disjunctive mandatory constraint is needed).



3. (c)

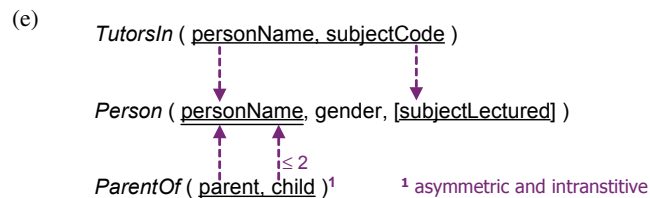


- (b) The involves predicate in the original conceptual schema and its external uniqueness constraint fail to cater for programmers changing their class *during* a project. An unrealistic resolution would be to demand that the external constraint be enforced anyway (e.g. by changing one or both of the programmers, and only storing the current project team); but this would fail anyway if the external uniqueness is still applied *after* the project is over and a programmer on the project has his/her class changed.

In realistic business practice, we would not change a programmer during a project anyway except in rare cases (e.g. demoted because of incompetence to work on the project). So the original conceptual schema needs to be changed. There are various options (e.g. delete the external uniqueness constraint, apply it only at the initial formation of a project team, or maintain history of project team membership). This change also would also require a change to the optimized schema.

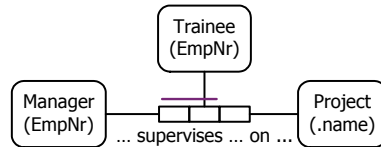
#### Exercise 14.6

1. (a) The Child column is not atomic. So the table is not even in 1NF.
- (b) The nonkey attribute is functionally dependent on just part of the key (person). hNF = 1.
- (c) Nonkey attributes depend on just part of the key (e.g. parent  $\rightarrow$  sexOfParent). hNF = 1.
- (d) Child attribute depends on only part of the key (tutorName).  
hNF = 3 (not 1, since child is part of the key).

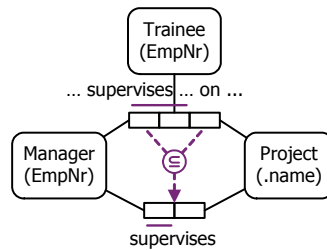


*Note:* If Subject plays other mandatory, functional roles (as is likely in practice) the fact type Person lectures Subject should be mapped instead to a Subject table that stores functional facts for Subject.

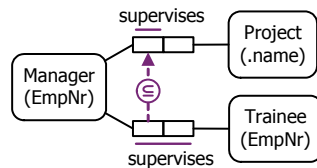
3. (a) This solution uses employee numbers instead of names, and ignores history aspects such as trainees later becoming managers.



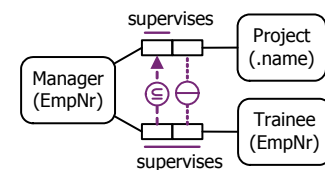
- (b) Here and later, we assume that this is just part of a schema in which managers play other mandatory roles (e.g. having a name), that it is possible for some managers to do no supervision, and that a project may have many supervisors.



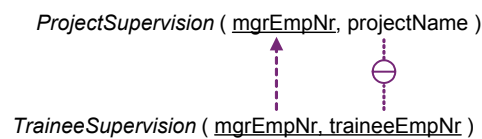
- (c) The ternary is compound, since there is an implied FD from its first role to its last role. So it should be split as shown below.



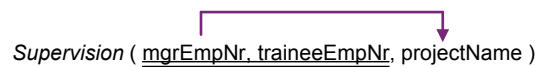
- (d)



- (e) This solution ignores all other fact types that might exist in the global schema.



- (f)



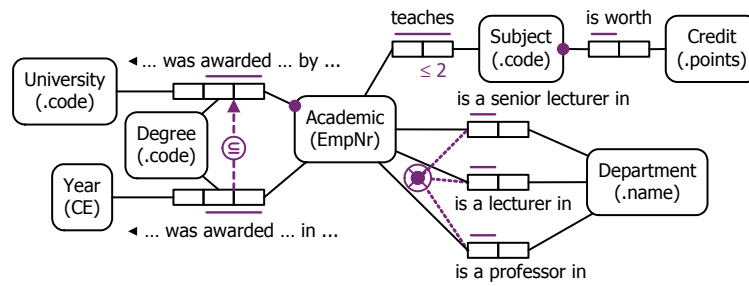
### Exercise 14.7

1.  $Student ( \underline{studentNr}, studentName, degree )$   
 $Result ( \underline{studentNr}, \underline{subjectCode}, studentname, [rating] )$

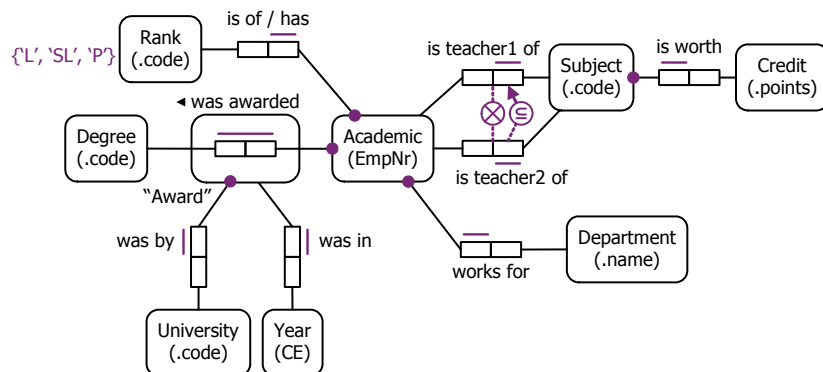
Minor disadvantages: updates are more expensive because of the pair-subset constraint; more storage space is required.

### Exercise 14.8

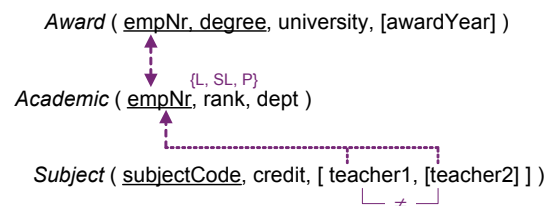
1. (a)



- (b)



1. (c)



1 (d)

**Award:**

| <i>empNr</i> | <i>degree</i> | <i>university</i> | <i>awardYear</i> |
|--------------|---------------|-------------------|------------------|
| 30572        | PhD           | UQ                | 1990             |

**Academic:**

| <i>empNr</i> | <i>rank</i> | <i>dept</i>      |
|--------------|-------------|------------------|
| 30572        | SL          | Computer Science |

**Subject:**

| <i>subjectCode</i> | <i>credit</i> | <i>teacher1</i> | <i>teacher2</i> |
|--------------------|---------------|-----------------|-----------------|
| CS115              | 8             | 30572           | ?               |

### Exercise 14.9

- 1 (a) The *Car* table is denormalized. Its effective key is *vin*. The non-key attributes *bwd*, *fwd*, “4wd”, *awd*, *manufacturer*, *modelName* and *yearmade* are all functionally dependent on the nonkey attribute *carModelId*. So facts such as CarModel ‘M1’ has Manufacturer ‘Mazda’ may be duplicated.

The *Color* table is denormalized. It has no effective key; even (*colorCode*, *primaryColor*) is not a key since *primaryColor* is optional. The attribute *colorName* is functionally dependent on *colorCode*, which is not a whole key. So facts such as Color ‘PL’ has ColorName ‘PLUM’ may be duplicated.

The *Customer* table is denormalized. It has no effective key; even (*customerId*, *phoneNr*) is not a key since *phoneNr* is optional. The nonkey attribute *customerName* is functionally dependent on the nonkey attribute *customerId*, and is combined with the multi-valued dependency from *customerId* to *phoneNr*. So facts such as Customer 201 has CustomerName ‘John Smith’ may be duplicated.

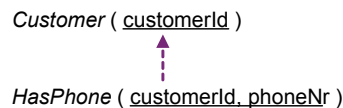
The *Sale* table is denormalized. The nonkey attribute *empName* is functionally dependent on the nonkey attribute *empNr*. So facts such as Employee 51 has EmpName ‘Tom Brown’ may be duplicated.

#### Comment

In the relational model, relations must have keys, and keys cannot have nullable attributes. In SQL, relations are not required to have keys at all. Consider the following simplified customer subschema.



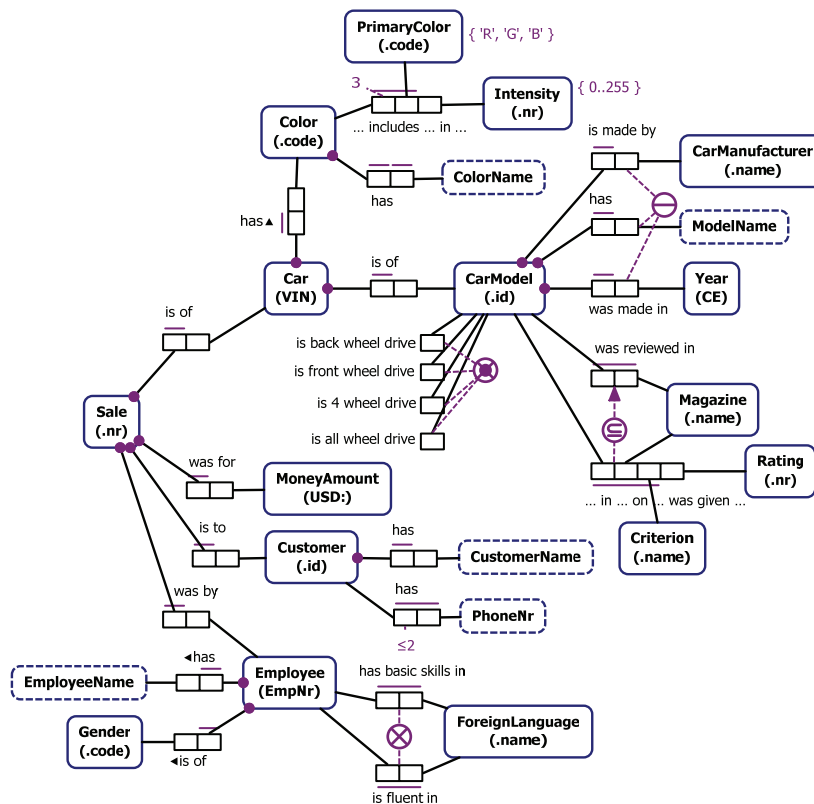
This Rmaps to the following relational schema, which is consistent with the relational model:



However, in SQL it is permitted to map this to a single, keyless table, as shown below. This table has a uniqueness constraint but no key, since keys cannot have nullable components. Although some people use the term “key” in a weaker sense that does allow nulls, this contradicts the definition of key in the relational model.

*Customer* ( customerId, [phoneNr] )

1. (b) The following schema was produced with the NORMA tool. There is an additional textual constraint that each RGB intensity pattern corresponds to at most one color.



- (c) **select** Review.carModelId, Review.magazine, criterion, rating  
**from** CarSale1.Review **left outer join** CarSale1.Result  
**on** Review.carModelId = Result.carModelId  
**and** Review.magazine = Result.magazine  
**and** criterion = 'Economy' -- *alter*: **where** criterion = 'Economy' or criterion is null
- (d) **select** Employee.empNr, max(Employee.empName) **as** empName, max(gender) **as** gender,  
 sum(salePrice) **as** salesTotal  
**from** CarSale1.Employee **join** CarSale1.Sale  
**on** Employee.empNr = Sale.empNr  
**where** Employee.empNr in

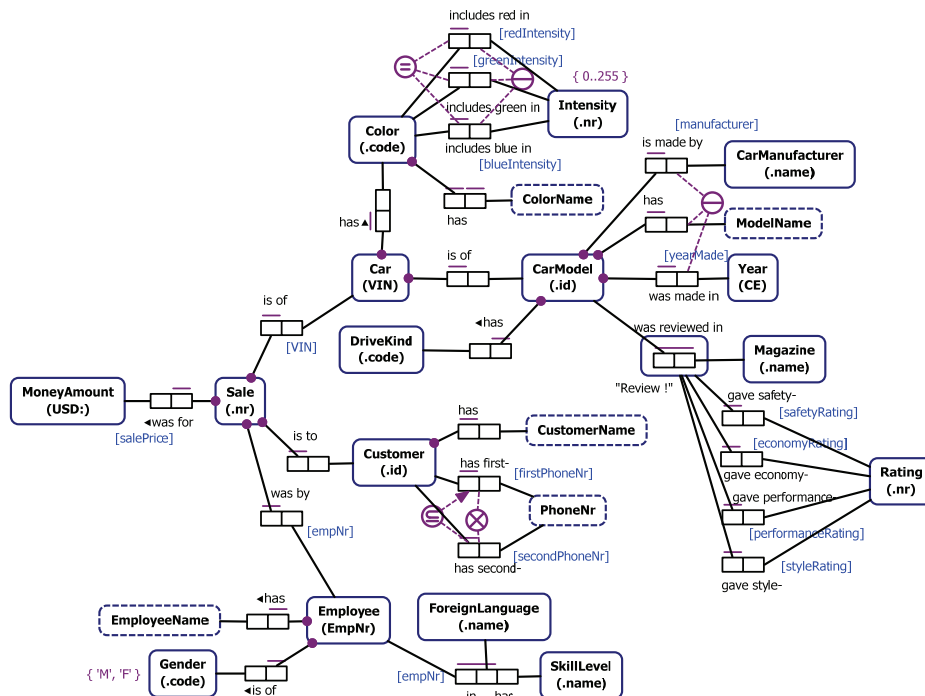
```

(select empNr from CarSale1.LanguageNovice
 where foreignLanguage = 'Spanish')
or Employee.empNr in
(select empNr from CarSale1.LanguageExpert
 where foreignLanguage = 'Spanish') -- aliter: empNr in Union
group by Employee.empNr -- aliter: group by name and gender too instead of using max

```

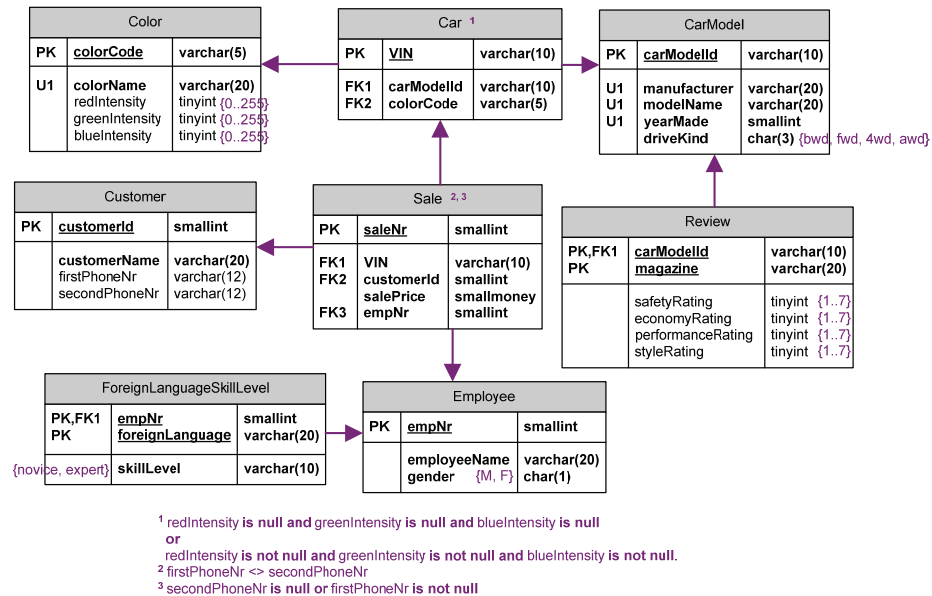
1. (e) **select** vin, Car.colorCode, carModelId,  
**case**  
    **when** bwd = 1 **then** 'bwd'  
    **when** fwd = 1 **then** 'fwd'  
    **when** "4wd" = 1 **then** '4wd'  
    **when** awd = 1 **then** 'awd'  
**end as** drive\_status  
**from** CarSale1.Car **join** CarSale1.Color **as** Color1  
    **on** Car.colorCode = Color1.colorCode  
**join** CarSale1.Color **as** Color2  
    **on** Color1.colorCode = Color2.colorCode  
    **and** Color1.primaryColor = 'R' **and** Color2.primaryColor = 'G'  
    **and** Color1.intensity = Color2.intensity

(f) This ORM schema was produced in the NORMA tool.





1. (g) The following relational schema was generated from the ORM Source Model solution in Microsoft Visio for Enterprise Architects.



- (h) The following trigger code is for SQL Server 2005.

```
create trigger tr_Color_RGBpatternUC
on CarSale2.Color
after insert, update as
if @@RowCount = 0 return
if Update(redIntensity) or Update(greenIntensity) or Update(blueIntensity)
if exists (select 1 from CarSale2.Color
 where redIntensity is not null and greenIntensity is not null and blueIntensity is
 not null
 group by redIntensity, greenIntensity, blueIntensity
 having count(*) > 1)
begin
 Raiserror ('Each RGB intensity pattern applies to at most one Color.',16,1)
 rollback transaction
end
```

- (d) The following code is written for SQL Server 2005.

```
insert into CarSale2.Color
select Rcolor.colorCode, Rcolor.colorName,
 Rcolor.intensity as redIntensity, Gcolor.intensity as greenIntensity, Bcolor.intensity as blueIntensity
from CarSale1.Color as Rcolor
join CarSale1.Color as Gcolor
```

```
on Rcolor.colorCode = Gcolor.colorCode
 and (Rcolor.primaryColor = 'R' or Rcolor.primaryColor is null)
 and (Gcolor.primaryColor = 'G' or Gcolor.primaryColor is null)
join CarSale1.Color as Bcolor on Gcolor.colorCode = Bcolor.colorCode
 and (Bcolor.primaryColor = 'B' or Bcolor.primaryColor is null)
```

```
insert into CarSale2.CarModel
select distinct carModelId, manufacturer, modelName, yearMade,
case
 when bwd = 1 then 'bwd'
 when fwd = 1 then 'fwd'
 when "4wd" = 1 then '4wd'
 when awd = 1 then 'awd'
end as driveKind
from CarSale1.Car
```

```
insert into CarSale2.Car
select VIN, carModelId, colorCode
from CarSale1.Car
```

```
insert into CarSale2.Review
select Review.carModelId, Review.magazine, SafetyReview.rating as safetyRating,
 EconomyReview.rating as economyRating, PerformanceReview.rating as performanceRating,
 StyleReview.rating as styleRating
from CarSale1.Review left outer join CarSale1.ReviewResult as SafetyReview
 on Review.carModelId = SafetyReview.carModelId and Review.magazine = SafetyReview.magazine
 and SafetyReview.criterion = 'Safety'
left outer join CarSale1.ReviewResult as EconomyReview
 on Review.carModelId = EconomyReview.carModelId
 and Review.magazine = EconomyReview.magazine and EconomyReview.criterion = 'Economy'
left outer join CarSale1.ReviewResult as PerformanceReview
 on Review.carModelId = PerformanceReview.carModelId
 and Review.magazine = PerformanceReview.magazine
 and PerformanceReview.criterion = 'Performance'
left outer join CarSale1.ReviewResult as StyleReview
 on Review.carModelId = StyleReview.carModelId and Review.magazine = StyleReview.magazine
 and StyleReview.criterion = 'Style'
```

```
insert into CarSale2.Customer
select Customer1.customerId, Customer1.customerName, Customer1.phoneNr, Customer2.phoneNr
from CarSale1.Customer as Customer1 left outer join CarSale1.Customer as Customer2
 on Customer1.customerId = Customer2.customerId and Customer1.phoneNr <> Customer2.phoneNr
where Customer1.phoneNr < Customer2.phoneNr
 or Customer2.phoneNr is null
```

*aliter:*

```
insert into CarSale2.Customer
select Customer1.customerId, Customer1.customerName, Customer1.phoneNr as firstPhoneNr,
 Customer2.phoneNr as secondPhoneNr
```

```
from CarSale1.Customer as Customer1 join CarSale1.Customer as Customer2
 on Customer1.customerId = Customer2.customerId
where Customer1.phoneNr < Customer2.phoneNr
 or Customer1.phoneNr is null and Customer2.phoneNr is null
union
select Customer.customerId, Customer.customerName, Customer.phoneNr, null
from CarSale1.Customer
where customerId in
 (select customerId from CarSale1.Customer
 group by customerId, customerName
 having count (distinct phoneNr) = 1)
```

```
insert into CarSale2.Employee
select * from CarSale1.Employee
```

```
insert into CarSale2.Sale
select saleNr, VIN, customerId, salePrice, empNr
from CarSale1.Sale
```

```
insert into CarSale2.ForeignLanguageSkillLevel
select *, 'novice' as skillLevel
from CarSale1.LanguageNovice
union
select *, 'expert' as skillLevel
from CarSale1.LanguageExpert
```

1. (j)

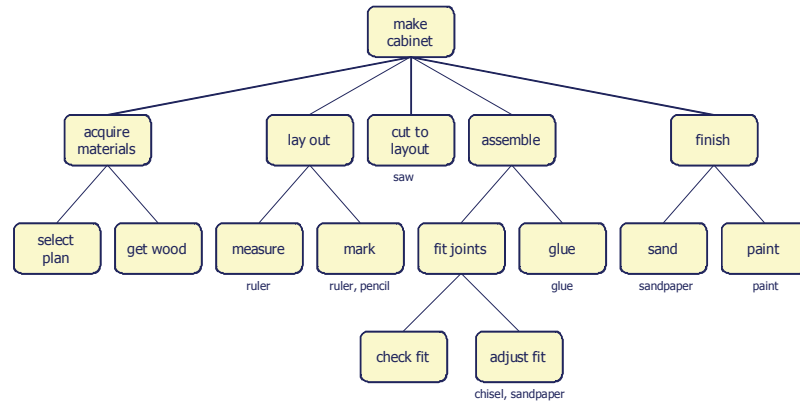
```
select carModelId, magazine, economyRating
from CarSale2.Review
```

```
select Employee.empNr, employeeName, gender, sum(salePrice) as salesTotal
from CarSale2.Employee join CarSale2.ForeignLanguageSkillLevel
 on Employee.empNr = ForeignLanguageSkillLevel.empNr
 and foreignLanguage = 'Spanish'
join CarSale2.Sale
 on Employee.empNr = Sale.empNr
group by Employee.empNr, employeeName, gender
```

```
select VIN, Car.colorCode, Car.carModelId, driveKind
from CarSale2.Car join CarSale2.Color
 on Car.colorCode = Color.colorCode
 and redIntensity = greenIntensity
join CarSale2.CarModel
 on Car.carModelId = CarModel.carModelId
```

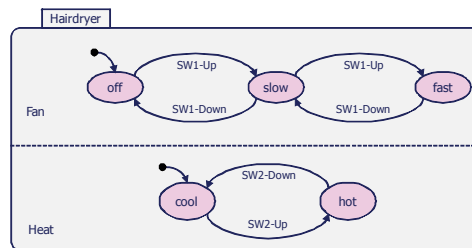
**Exercise 15.7**

1. (a)

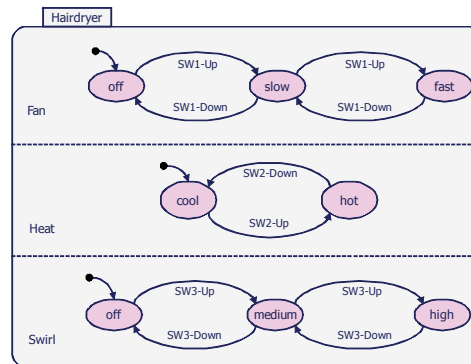


- (b) The resources that appear to be appropriate are marked on the diagram above. Hmmm – we don't seem to have any use for the screwdriver.
- (c) Some resources are consumed by activities. Examples in this case would be glue, sandpaper and paint. We would have to monitor the consumption of these and replenish at frequent intervals. Other resources need to be there, but are not consumed by the activities (except in a microscopic way). Examples in this case are saw, ruler, chisel.
- (d) The process should be suspended after gluing and before finishing to give the glue time to set. Also, note that the process cannot be considered complete until the paint has dried, even though there are no subsequent activities
- (e) The activities “layout” and “cut to layout” will be required for each part. The activity “assemble” will be required for each set of joining parts. We can't predict in advance how many instances of these activities there will be (why?). The activities “acquire materials” and “finish” would probably only occur once per process instance. (Given sufficient resources we could also have multiple instances of the whole “make cabinet” process active simultaneously.)
- (f) This is a trivial example, but it reveals many of the features of real business process design
  - we start from a business description
  - we break the process down into activities that we can define in a straightforward way, usually a hierarchical breakdown
  - we associate resource types with each activity
  - defining the process may make us realize that we need to define or refine other elements of our business model
    - other processes (such as replenish glue, paint and sandpaper)
    - other types of element (such as rules to specify the fit of the joints)
    - more data (e.g., the properties of the wood) that might be required by the process
  - note that the structure is not sufficient to completely define the process (e.g., think about the “cut to layout” and “assemble” activities).

3. (a) Although the state model is consistent with the description given, the state  $\langle \text{off}, \text{hot} \rangle$  does not seem to be very sensible – we could predict a rapid melt-down of the hair dryer under these conditions! Problems of this kind could be caused by sloppy specification or a genuine design flaw. An information analyst has the responsibility to identify such issues and to facilitate their resolution.
- (b) A composite diagram would resemble the following.



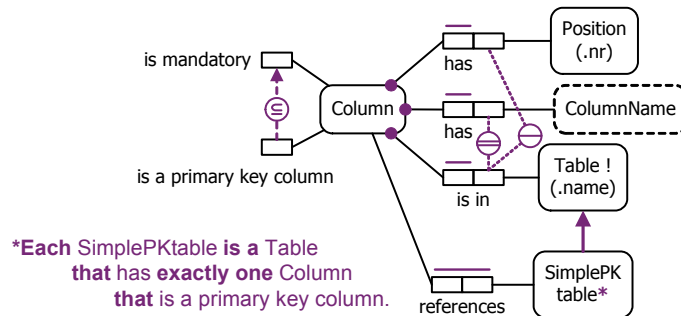
- (c) A composite diagram with the “Swirl” control would resemble the following.



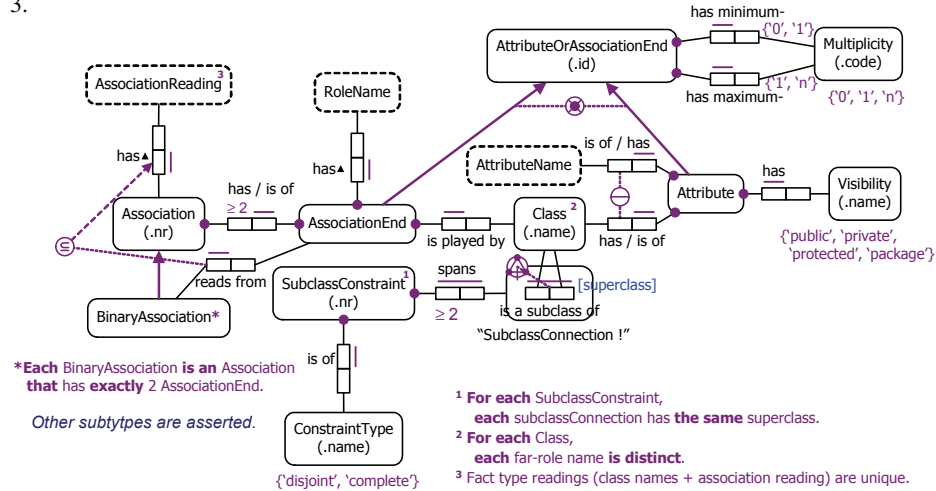
- (d) Without composite states we are faced with a potential explosion of replication. In this case, we would need the equivalent of three copies of the original six-state diagram; one copy for each of the “swirl” positions. In realistic scenarios, with many attendant variables, state diagrams quickly become unmanageable if composite states are not used.
5. The two workflows are equivalent. The workflow on the left contains arbitrary cycles, so, for example, it’s possible to jump into the middle of a looping construct. In general, arbitrary loops can be converted to a block structured form, though this may take a little thought. The workflow on the right includes some auxiliary variables ( $y$  and  $z$ ) that are specified in terms of original variables  $w$  and  $x$ . The main loop (a block) is controlled by  $z$  – one of the auxiliary variables. This kind of transformation may be necessary when dealing with languages that are purely block structured.

### Exercise 16.8

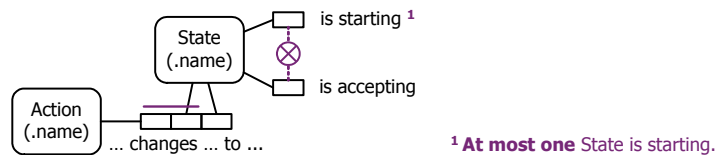
1. This allows creation of a table before adding columns.



- 3.



- 5.



Note: A completed transition graph has additional constraints:

- At least one State is starting.**
- At least one State is accepting.**
- For each State<sub>1</sub>,**
  - some Action changes State<sub>1</sub> to some State<sub>2</sub>**
  - or some Action changes some State<sub>2</sub> to State<sub>1</sub>.** (inclusive-or constraint on State's roles in the ternary).
- The graph is connected** (no gaps)

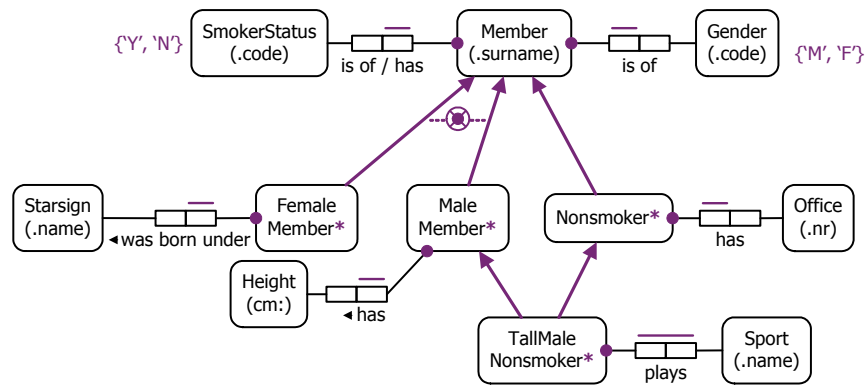
**Exercise A1** (Appendix A)

1. B C F G I

3. “709T” sounds like “seven oh ninety”. Consequently most customers wrote it as “7090”. So IBM renamed it to agree with common practice.

**Exercise B1** (Appendix B)

1.



\*Each FemaleMember is a Member who is of Gender 'F'.

\*Each MaleMember is a Member who is of Gender 'M'.

\*Each Nonsmoker is a Member who has SmokerStatus 'N'.

\*Each TallMaleNonsmoker is a Nonsmoker and is a MaleMember who has Height >= 180.

**Exercise C1** (Appendix C)

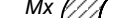
1. (a) **select** person **from** Applicant  
**where** IQ > 120  
**and** person **in**  
     (select person **from** Speaks  
     **where** "language" = 'English')  
**and** person **in**  
     (select person **from** Speaks  
     **where** "language" = 'Japanese')
- (b) **select distinct** person **from** Speaks  
**where** "language" **in**  
     (select "language" **from** Speaks  
     **where** person = 'Fumie')
- (c) **select distinct** person **from** Speaks X  
**where not exists**  
     (select "language" **from** Speaks  
     **where** person = 'Chris'  
     **and** "language" **not in**  
         (select "language" **from** Speaks  
         **where** person = X.person))

1. (d) **select max(person)**  
**from Speaks**  
**group by "language"**  
**having count(\*) = 1**  
*aliter:* **select person from Speaks X**  
**where 1 =**  
           **(select count(\*) from Speaks**  
           **where "language" = X."language")**
- (e) **select distinct person from Speaks X**  
**where not exists**  
       **(select "language" from Speaks**  
       **where person = X.person**  
       **and "language" in**  
           **(select "language" from Speaks**  
           **where person = 'Chris'))**
- (f) **select distinct person from Speaks X**  
**where not exists**  
       **(select "language" from Speaks**  
       **where person = X.person**  
       **and "language" not in**  
           **(select "language" from Speaks**  
           **where person = 'David'))**
- (g) **select person from Speaks**  
**group by person**  
**having count(\*) =**  
       **(select count(distinct "language" from Speaks)**
- (h) **select person from Speaks**  
**group by person**  
**having count(\*) >= 3**
- (i) **select distinct person from Speaks X**  
**where person <> 'David'**  
**and not exists**  
       **(select "language" from Speaks**  
       **where person = X.person**  
       **and "language" not in**  
           **(select "language" from Speaks**  
           **where person = 'David')**  
       **or person = 'David'**  
       **and "language" not in**  
           **(select "language" from Speaks**  
           **where person = X.person))**
- (j) **select person from Applicant X**  
**where IQ = (select max(IQ) from Applicant**  
       **where sex = X.sex)**



1. (k) **select distinct "language" from Speaks X  
where not exists**  
          **(select person from Speaks**  
          **where person in**  
                  **(select person from Applicant**  
                  **where sex = 'M')**  
          **and person not in**  
                  **(select person from Speaks**  
                  **where "language" = X."language"))**
- (l) g            (m) none            (n) b            (o) d            (p) I
- (q) f            (r) none            (s) e            (t) c

3. C

5.   $Mx = \text{Members of team } x$   
 $My = \text{Members of team } y$
- $Mx = My$   $\#(Mx \cap My) = \#Mx$   
 $= \#My$

**check not exists**

```
(select 'extensional uniqueness constraint violation'
from Membership as X, Membership as Y
where X.team < Y.team
 and X.member = Y.member
group by X.team, Y.team
having count(*) =
 (select count(*) from Membership
 where team = X.team)
and count(*) =
 (select count(*) from Membership
 where team = Y.team))
```

### Exercise D1 (Appendix D)

1. (a) **select** person  
**from** Speaks  
**group by** person  
**having** count( \*) <= all  
    (select count( \*)  
      **from** Speaks  
      **group by** person)
- (b) **select** person  
**from** Speaks  
**group by** person  
**having** count( \*) >= all

```
(select count(*)
 from Speaks
 group by person)
```

1. (c) **select** gender  
      **from** Applicant  
      **group by** gender  
      **having** avg( IQ ) >= **all**  
          ( **select** avg( IQ )  
            **from** Applicant  
            **group by** gender)
- (d) **select** gender  
      **from** Applicant  
      **group by** gender  
      **having** max( iq ) - min( iq ) >= **all**  
              ( **select** max( iq ) - min( iq )  
                **from** Applicant  
                **group by** gender)

or alternatively:

```
select top 1 a1.gender
from Applicant as a1 join Applicant as a2
on a1.gender = a2.gender
 and a1.iq > a2.iq
order by a1.iq-a2.iq desc
```

3. (a) **select top 5** productId, productName, unitsInStock  
      **from** Product  
      **order by** unitsInStock **desc**
- (b) **select top 5 with ties** productId, productName, unitsInStock  
      **from** Product  
      **order by** unitsInStock **desc**
- (c) **select** productId, productName, unitPrice  
      **from** Product  
      **where** unitPrice **in**  
          ( **select top 5** unitPrice **from** Product  
            **order by** unitPrice **desc** )  
      **order by** unitPrice **desc**
- (d) **select top 10 percent** productId, productName, unitsInStock  
      **from** Product  
      **where** unitsInStock > 0  
      **order by** unitsInStock **desc**

3. (e) **select** productId, productName, unitsInStock,  
    **row\_number()** **over** ( **order by** unitsInStock desc ) **as** row,  
    **rank()** **over** ( **order by** unitsInStock desc ) **as** rank,  
    **dense\_rank()** **over** ( **order by** unitsInStock desc ) **as** dense\_rank  
**from** Product  
**where** unitsInStock > 100  
**order by** unitsInStock desc

**Exercise E1** (Appendix E)

1. (a) **create view** StudentScore **as**  
    **select** studentNr, studentName, testScore, examScore,  
        ( testScore + examScore ) **as** totalScore **from** Student
- (b) **with** studentScore **as**  
    ( **select** studentNr, studentName, testScore, examScore,  
        ( testScore + examScore ) **as** totalScore **from** Student )  
**select** \* **from** studentScore  
**where** totalScore < 50
- (c) **create table** Student2 (  
    studentNr **int not null primary key**,  
    studentName **varchar**( 50 ) **not null**,  
    testScore **int not null**,  
    examScore **int not null**,  
    totalScore **as** testScore + examScore ) -- SQL Server syntax