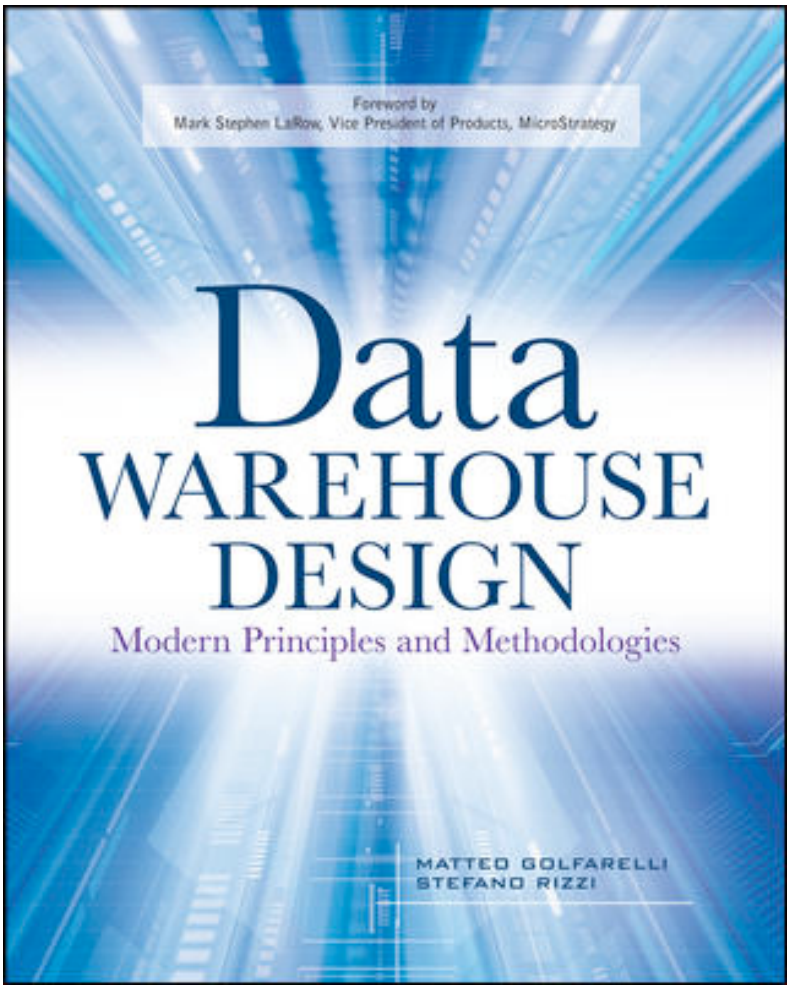


Exercises

Matteo Golfarelli, Stefano Rizzi

Part 1:
ANALYSIS AND INTEGRATION OF DATA SOURCES

❑ Exercise 1.1

Two operational database schemata are given:

RESEARCH_DB

```
AUTHORS(AuthCode,FirstName,LastName,Position,IdNumber)
ARTICLES(ArtCode,Title,MainAuthor:AUTHORS,JournalName,JournalPublisher)
```

NEW_RESEARCH_DB

```
RESEARCHERS(IdNumber,ResearcherFirstName,ResearcherLastName,Position,
  RecruitmentDate)
JOURNALS(JournalCode,JournalName,Language,Classification,PublisherCode,
  PublisherName,PublisherCountry)
RESEARCH_PRODUCTS(ProdCode,Title,Type,JournalCode:JOURNALS)
// research products may be of type "journal paper" or "patent"
AUTHOR_PRODUCT(ProdCode:RESEARCH_PRODUCTS,IdNumber:RESEARCHERS,IsMain)
```

The first database stores the articles written by researchers of the University of Bologna till 2000; the second database stores the articles and the patents since 2000 (a patent has one or more authors but it is not associated to any journal; the `IsMain` attribute is Boolean). Attributes with the same name in different schemata are assumed to have the same meaning.

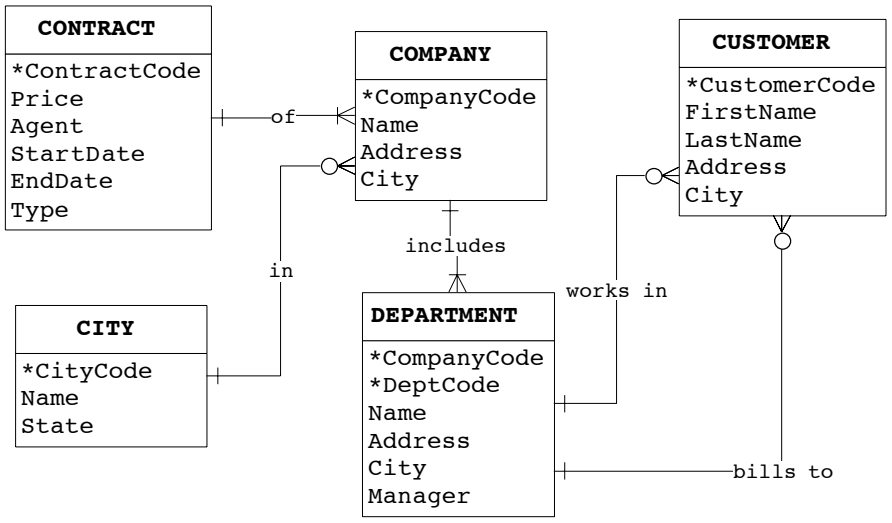
Inspect and normalize both databases, making reasonable assumptions on the multiplicity of relationships, then integrate them to draw a reconciled Entity-Relationship schema.

❑ Exercise 1.2

A help desk hinges on the concept of *call*, meant as a contact made by a customer to report a problem or a comment about a product he bought. The help-desk service includes a front-end where operators take the customer calls, and a back-end where some experts try to solve the problems. The customer opens the contact by calling the front-end. If possible, the front-end directly manages the problem and gives a solution to the customer, otherwise he hands the problem to an expert. During its lifetime, a call can be overtaken by different persons and groups, but in the end its solution will always be communicated to the customer by a front-end operator. Every activity carried out during a call gives rise to a call detail, so the story of a call can be reconstructed by orderly browsing its details. The problems managed by the help desk are classified into a recursive subject hierarchy.

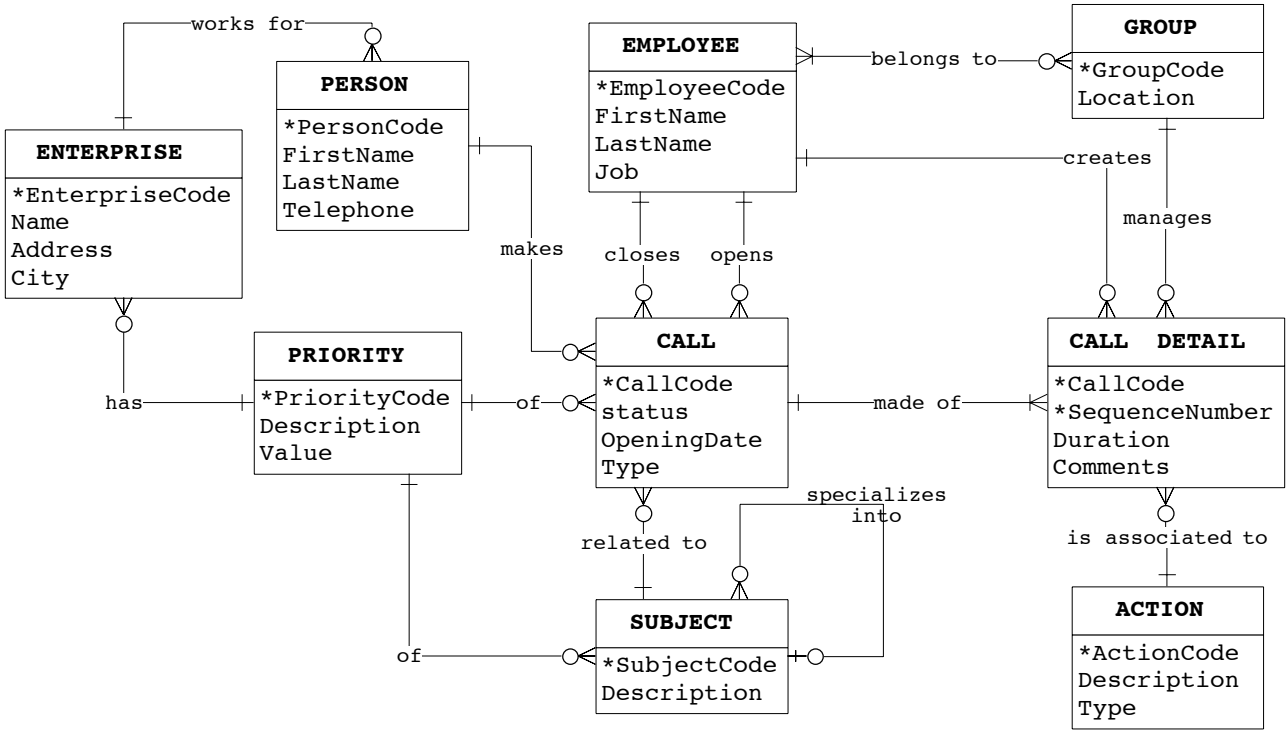
CUSTOMERS_DB

For every customer, the department where he works and the one he bills to are distinguished.



CALLS_DB

The unary association among subjects models a tree hierarchy. For every call, the employee who opens it and the one who closes it are modeled. All the passages taking place between the call opening and closing are modeled by actions associated to call details. Each call detail is associated to the employee and the group that manage it. Subjects, as well as calls and enterprises, have a priority; the priority of a call is not necessarily determined by the one of its subject or enterprise.



Inspect and normalize both databases, then integrate them to draw a reconciled schema.

❑ Exercise 1.3

Two relational schemata are given. The first one describes the highway tolls for employee-operated tollgates. The second one has been created later on to manage also ATM and cash automatic tollgates.

DB1

```
GATES(GateId,GateName,State)
DISTANCES(FromGate:GATES,ToGate:GATES,DistanceRange,Charge)
TOLLS(FromGate:GATES,ToGate:GATES,TimeIn,DateIn,TimeOut,DateOut,Charge)
```

DB2

```
GATE_TYPES(TypeId,Description)
GATES(GateId,GateName,Highway,County,State,TypeId:GATE_TYPES)
DISTANCES(FromGate:GATES,ToGate:GATES,DistanceInKm,RangeId:DISTANCE_RANGES)
DISTANCE_RANGES(RangeId,FromKms,ToKms)
TARIFF(RangeId:DISTANCE_RANGES,Charge)
TOLLS(TollId,FromGate:GATES,ToGate:GATES,TimeIn,DateIn,TimeOut,DateOut)
```

Inspect and normalize both databases, then integrate them to draw a reconciled relational schema.

❑ Exercise 1.4

Two relational schemata in the finance area are given:

STOCK_DB

```
SHARES(ShareCode,Name,Type)
RATINGS(ShareCode:SHARES,Date,Price)
PERSONS(SSN,FirstName,LastName)
PERSONS_OWN_SHARES(ShareCode:SHARES,SSN:PERSONS,Quantity)
```

CUSTOMER_DB

```
CUSTOMERS(CustomerCode,FirstName,LastName,BirthDate,Sex,BirthPlace)
PORTFOLIO(CustomerCode:CUSTOMERS,OpeningDate,TotalValue)
STOCK(StockCode,Name,Type)
TRANSACTIONS(ProgressiveId,StockCode:STOCK,CustomerCode:CUSTOMERS,Date,Quantity)
```

Inspect and normalize both databases, then integrate them to draw a reconciled Entity-Relationship schema.

Part 2:
CONCEPTUAL DESIGN

❑ Exercise 2.1

The following logical schema describes an operational database for credit card payments:

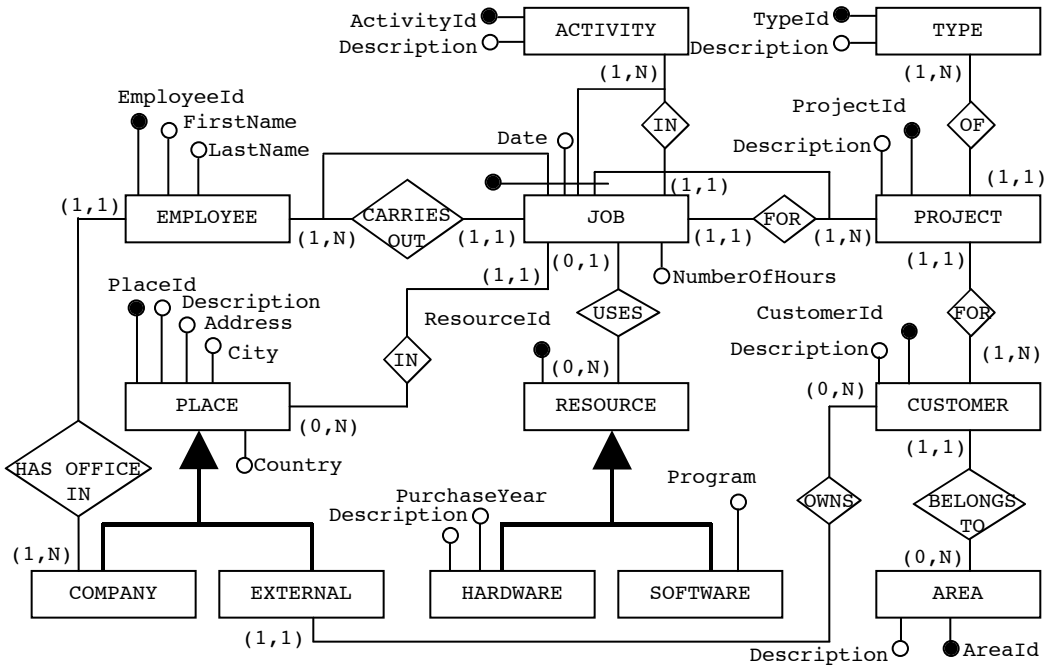
```
PURCHASES(PurchCode,Date,Amount,Currency,Exchange,CCNumber:CREDIT_CARDS,
     StoreCode:STORES)
CREDIT_CARDS(CCNumber,TypeCode:CARD_TYPES,HolderCode:HOLDERS,ExpirationDate,
     CreditLimit)
HOLDERS(HolderCode,FirstName,LastName,BirthDate,Job,IncomeRange,Quality)
CARD_TYPES(TypeCode,Network,IssuingBank,Address,Country,CardName,Color)
STORES(StoreCode,Name,Address,Country,Area,WebSite,Email,Type)
```

Some hidden functional dependencies hold: `Address→Country→Area`; `Job,IncomeRange→ Quality`; `IssuingBank→Address`.

Inspect and normalize the source schema, then choose a fact of interest and design its fact schema.

❑ Exercise 2.2

The following Entity-Relationship schema describes an operational database for a company's projects. The hidden functional dependency `Address→City→Country` holds.



Inspect the source schema, then choose a fact of interest and design its fact schema.

❑ Exercise 2.3

The following logical schema describes an operational database for motor races:

```
RACING_CIRCUITS(CircuitName,City,Length,Description)
GRAND_PRIXS(Date,CircuitName:RACING_CIRCUITS,NumberOfLaps)
DRIVERS(Name,Surname,BirthDate,Comments)
RACING_STABLES(StableName,MainAddress,Manager)
CARS(Number,StableName:RACING_STABLES)
DRIVES((Name,Surname):DRIVERS,Number:CARS,Year,FromDate,ToDate)
ARRIVALS((Date,CircuitName):GRAND_PRIXS,(Name,Surname,Number,Year):DRIVES,
     Position,Time)
SCORES(Year,Position,Score)
```

Choose a fact of interest and design its fact schema.

❑ Exercise 2.4

The following logical schema describes an operational database for research project fundings:

```
PROJECT_PROPOSALS(ProjectCode,Title,ProposalDate,SubjectCode:SUBJECTS,
     CoordinatorUnit:RESEARCH_UNITS)
RESEARCH_UNITS(Abbreviation,Name,Country)
OTHER_PROJECT_UNITS(ProjectCode:PROJECT_PROPOSALS,Abbreviation:RESEARCH_UNITS)
RESEARCHERS(ResearcherName,Country,BelongsTo:RESEARCH_UNITS)
AREAS(AreaCode,Area)
SUBJECTS(SubjectCode,Subject,AreaCode:AREAS)
PROJECT_FUNDINGS(ProjectCode:PROJECT_PROPOSALS,Amount,Funding_Date)
RESEARCH_UNITS_FUNDINGS(ProjectCode:PROJECT_FUNDINGS,MemberUnit:RESEARCH_UNITS,
     Amount)
```

A proposal is made from a number of research units, one of which is the coordinator. Not every proposal is approved and funded. The `RESEARCH_UNITS_FUNDINGS` relation describes how the total project funding is allotted among the participating units.

Inspect and normalize the source schema, then choose a fact of interest and design its fact schema.

❑ Exercise 2.5

The following logical schema describes an operational database for car rentals:

```
RENTAL_OFFICES(OfficeName,City,Area,State,Country)
CARS(LicensePlate,Category,Model,Brand,Fuel,RegistrationDate)
HAVE_OPTIONAL(LicensePlate:CARS,Optional)
RENTALS(LicensePlate:CARS,PickupDate,DropoffDate,
     PickupPlace:RENTAL_OFFICES,DropoffPlace:RENTAL_OFFICES,Miles)
DRIVERS(LicenseNumber,LicenseExpiration,DriverName,Birthdate)
DRIVE(LicenseNumber:DRIVERS,(LicensePlate,PickupDate):RENTALS)
INSURANCES(Risk,(LicensePlate,PickupDate):RENTALS,Cost)
PAYMENTS((LicensePlate,PickupDate):RENTALS,Amount,Discount,PaymentMode)
```

Some hidden functional dependencies hold: `City→State→Country→Area`, `Model→Brand`.

Inspect and normalize the source schema, then choose a fact of interest and design its fact schema.

❑ Exercise 2.6

The following logical schema describes an operational database for skiing areas:

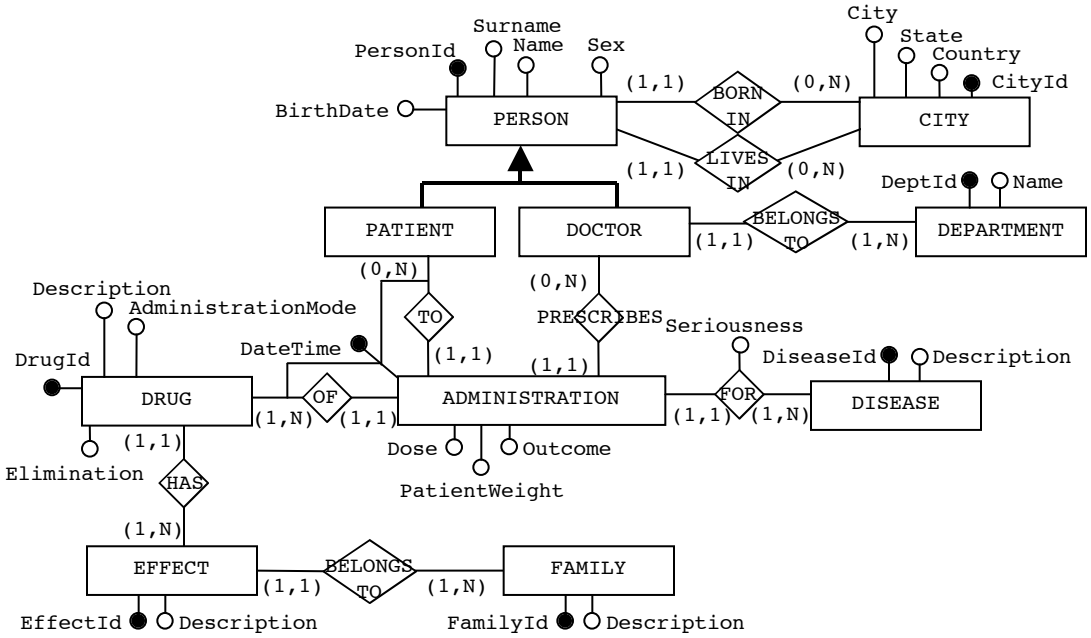
```
LIFT_SYSTEMS(LiftSystemCode,Type,Length,Duration,Points)
SKI_SLOPES(SlopeCode,Length,Colour)
LIFTS_FOR_SLOPES(SlopeCode:SKI_SLOPES,LiftSystemCode:LIFT_SYSTEMS)
SALE_OFFICES(OfficeCode,Name,Address)
SKI_PASSES(OfficeCode:SALES_OFFICES,PurchaseDate,ProgressiveNumber,PurchaseTime,
     TypeCode:SKI_PASS_TYPES,HolderName,Discount)
SKI_PASS_TYPES(TypeCode,Description,Days,Points,Cost)
TICKETS(TicketCode,PurchaseDate,OfficeCode:SALES_OFFICES,
     LiftSystemCode:LIFT_SYSTEMS,Cost)
LIFTS(LiftSystemCode:LIFT_SYSTEMS,Date,Time,TicketCode:TICKETS,
     (OfficeCode,PurchaseDate,ProgressiveNumber):SKI_PASSES)
```

Lift systems can be either cableways, chair-lifts, or ski-lifts; each lift system is worth a given number of points (`LIFT_SYSTEMS.Points`), that are taken off from point-based ski passes at each lift. A ski slope can be catered by several lift systems, and vice versa (`LIFTS_FOR_SLOPES` relation). Ski passes can be daily, weekly, point-based (20, 50, 100 points); only daily and weekly ski passes carry the skier name. `Discount` is a Boolean attribute. A skier can access a lift system either with a ski pass or by buying a single ticket for that lift system.

Inspect and normalize the source schema, then choose a fact of interest and design its fact schema.

❑ Exercise 2.7

The following Entity-Relationship schema describes an operational database for drug prescription and administration.



Inspect and normalize the source schema, then choose a fact of interest and design its fact schema.

Part 3:
LOGICAL DESIGN

❑ Exercise 3.1

Design a logical schema for the PURCHASE fact schema designed in Exercise 2.1.

❑ Exercise 3.2

Design a logical schema for the JOB fact schema designed in Exercise 2.2.

❑ Exercise 3.3

Design a logical schema for the ARRIVAL fact schema designed in Exercise 2.3.

❑ Exercise 3.4

Design a logical schema for the FUNDING fact schema designed in Exercise 2.4.

❑ Exercise 3.5

Design a logical schema for the RENTAL fact schema designed in Exercise 2.5.

❑ Exercise 3.6

Design a logical schema for the LIFT fact schema designed in Exercise 2.6.

❑ Exercise 3.7

Design a logical schema for the ADMINISTRATION fact schema designed in Exercise 2.7.

❑ Exercise 3.8

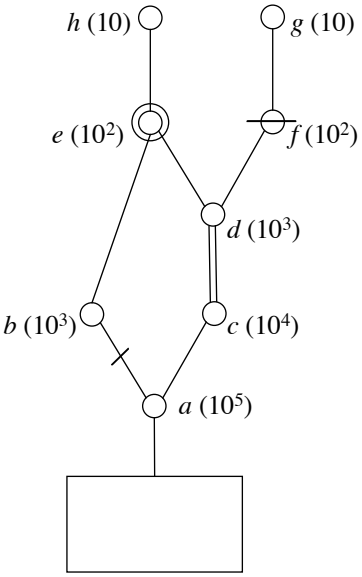
A PURCHASE fact schema is given that includes, among the others, three degenerate dimensions `sex`, `ageRange`, and `country` used to describe the buyer. The total number of primary events is about 100 000; the three degenerate dimensions take 1 byte, 5 bytes, and 20 bytes respectively; 5 age ranges and 30 countries are represented.

Find the logical design solution that minimizes the total space, assuming that each surrogate key takes 4 bytes.

❑ Exercise 3.9

In the DFM hierarchy below (in parentheses the attribute cardinalities), each attribute takes 20 bytes; every value of *c* is associated, on the average, with 3 distinct values of *d* through the multiple arc.

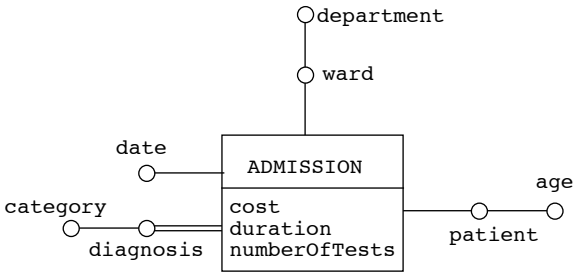
Carry out logical design of the hierarchy and estimate the space used by dimension tables.



❑ Exercise 3.10

The data volume for the ADMISSION fact schema is characterized as follows:

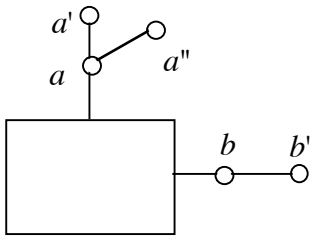
- all attributes and measures take 20 bytes;
- surrogate keys take 4 bytes;
- there are 20 wards, 100 diagnoses, 10 000 patients;
- there are about 50 admissions a day, each featuring 3 diagnoses in the average;
- the admission history covers 3 years.



Describe the two possible solutions for the multiple arc, and compute the total space for each of them.

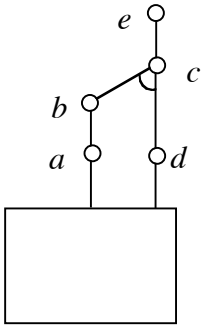
❑ Exercise 3.11

Given the fact schema below, draw its multidimensional lattice.



❑ Exercise 3.12

In the fact schema below, each attribute takes 100 bytes and the fact measures take 300 bytes as a whole. The fact cardinality is 100 000, and the other cardinalities are as follows: card(*a*)=1000, card(*b*)=500, card(*c*)=20, card(*d*)=50, card(*e*)=10.

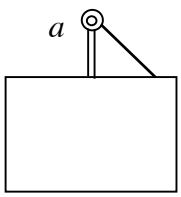


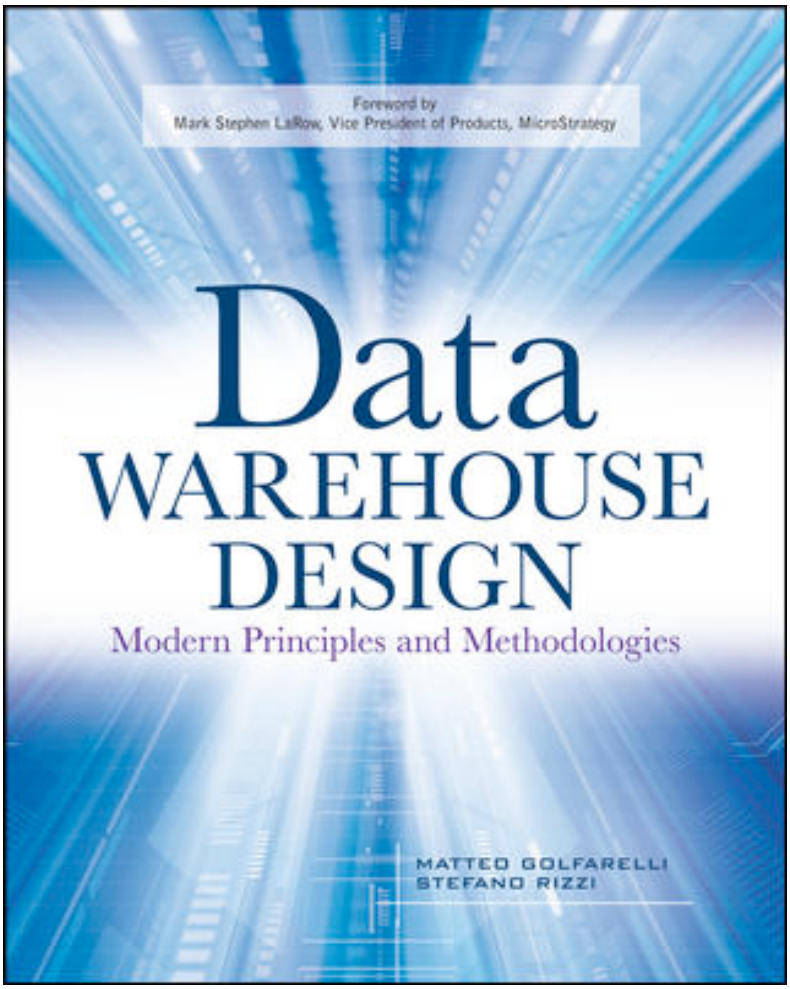
Find the possible logical design solutions and estimate their storage space.

❑ Exercise 3.13

In the fact schema below, attribute *a* takes 100 bytes and the fact measures take 300 bytes as a whole. The fact cardinality is 1 000 000, and *a* takes 1000 distinct values. Each primary event is associated with 3 values of *a* in the average.

Find the possible logical design solutions and estimate their storage space.





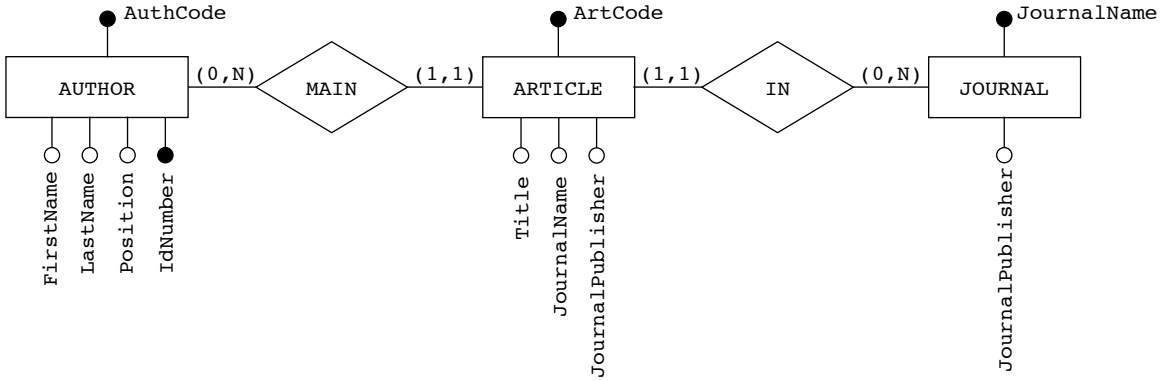
Solutions

Matteo Golfarelli, Stefano Rizzi

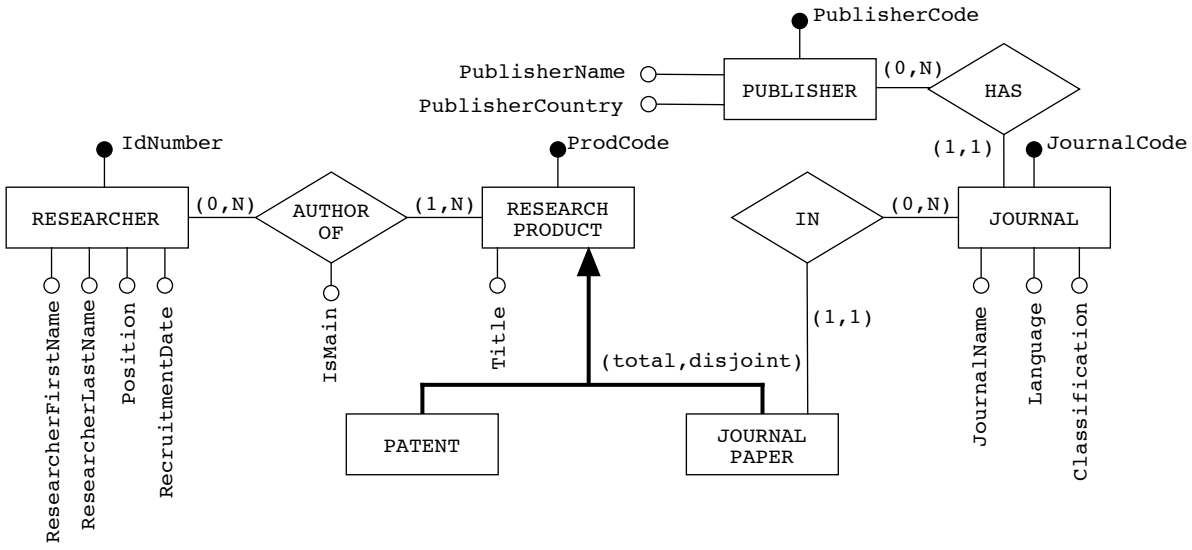
Part 1:
ANALYSIS AND INTEGRATION OF DATA SOURCES

✔ Exercise 1.1

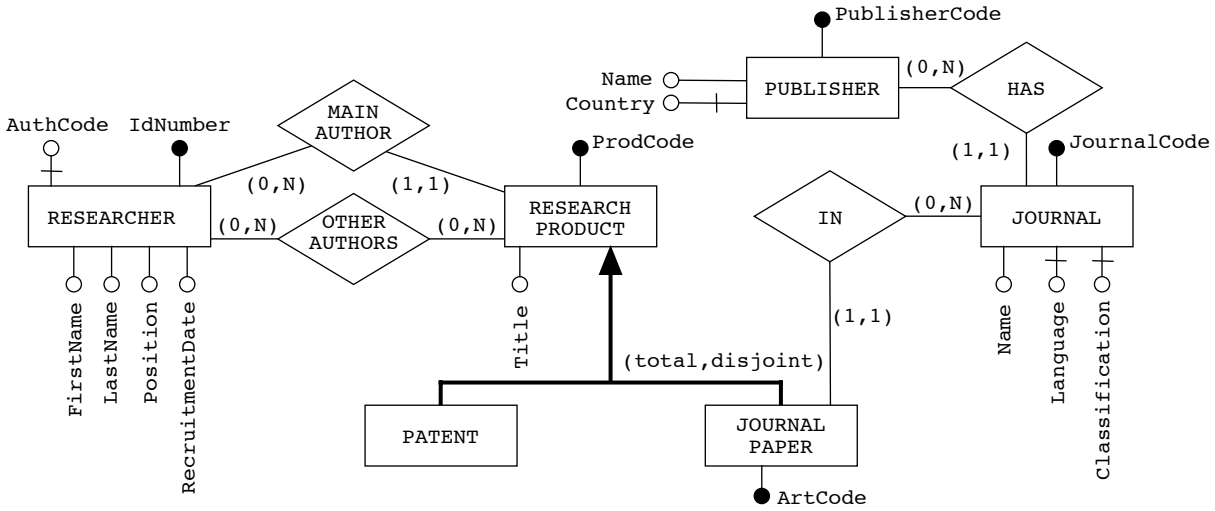
The results of an inspection and normalization of `RESEARCH_DB` are shown below. The `JOURNAL` entity has been added to normalize the `ARTICLES` relation, that is not in 3rd normal form due to the transitive functional dependency `ArtCode→JournalName→JournalPublisher`.



The Entity-Relationship schema for the second database is shown below. A specialization hierarchy has been added to represent that products can be either papers or patents, and that only papers have a journal. The `PUBLISHER` entity has been added to normalize `JOURNALS`, that is not in 3rd normal form.



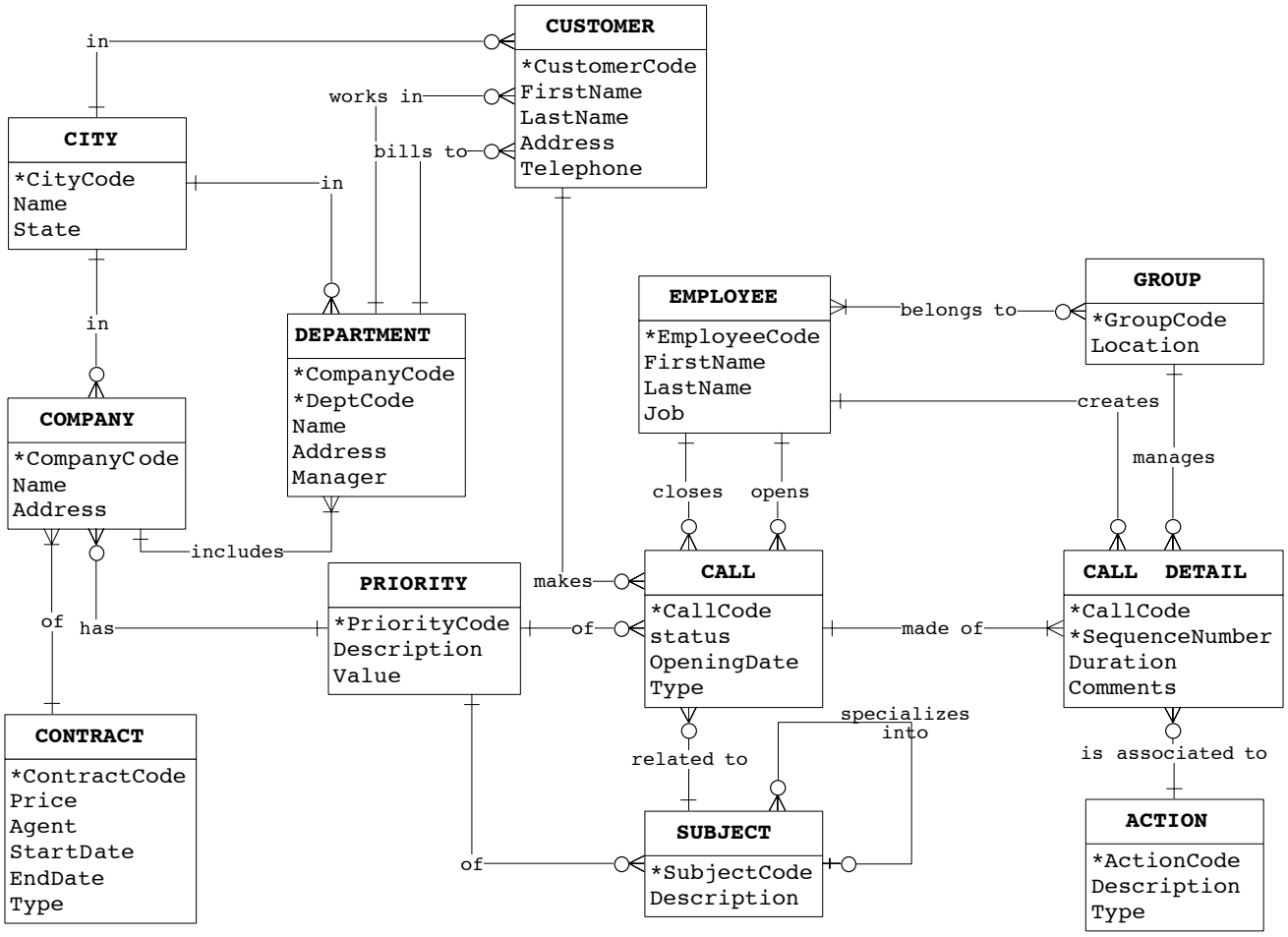
The reconciled schema integrates both sources schemata. The author-researcher synonymy has been solved by recognizing that "author" is actually a role of a researcher in the association with products. The `ARTICLE` entity in the first database matches the `JOURNAL PAPER` entity in the second database. To better represent the author-product association, two distinct relationships are introduced: `MAIN AUTHOR`, that models the main author of a product (integrating `RESEARCH_DB.ARTICLES.MainAuthor` and `NEW_RESEARCH_DB.AUTHOR_PRODUCT.IsMain`), and `OTHER AUTHORS`. Finally, note that `AuthCode` is not an identifier for `RESEARCHER` because its value is null for all articles before 2000.



✔ Exercise 1.2

In this case we have chosen to keep the Crow-Foot formalism for the reconciled schema. The main changes are as follows:

- The company-enterprise and customer-person synonymies have been solved.
- The denormalized `City` attributes have been removed and replaced by associations with `CITY`.
- The `works for` relationship in `CALLS_DB` has been recognized to be equivalent to the transitive composition of the `includes` and `works in` relationships in `CUSTOMERS_DB`.



✔ Exercise 1.3

Since in this case the target reconciled schema is required to take the same form of the source schemata, i.e., that of a relational schema, we will not use the Entity-Relationship formalism.
The first database suffers from two problems. Firstly, the key of the TOLLS relation is actually a superkey, because FromGate, TimeIn, and DateIn are sufficient to single out a toll. Secondly, Charge is redundant because, for every toll, it can be determined by accessing the DISTANCES relation. In the second database, GATES is not in 3rd normal form because County functionally determines State. Besides, after recognizing that there is a 1-to-1 association between TARIFF and DISTANCE_RANGES, we have decided to merge them. The two databases after normalization are shown below.

DB1

```
GATES(GateId,GateName,State)
DISTANCES(FromGate:GATES,ToGate:GATES,DistanceRange,Charge)
TOLLS((FromGate,ToGate):DISTANCES,TimeIn,DateIn,TimeOut,DateOut)
```

DB2

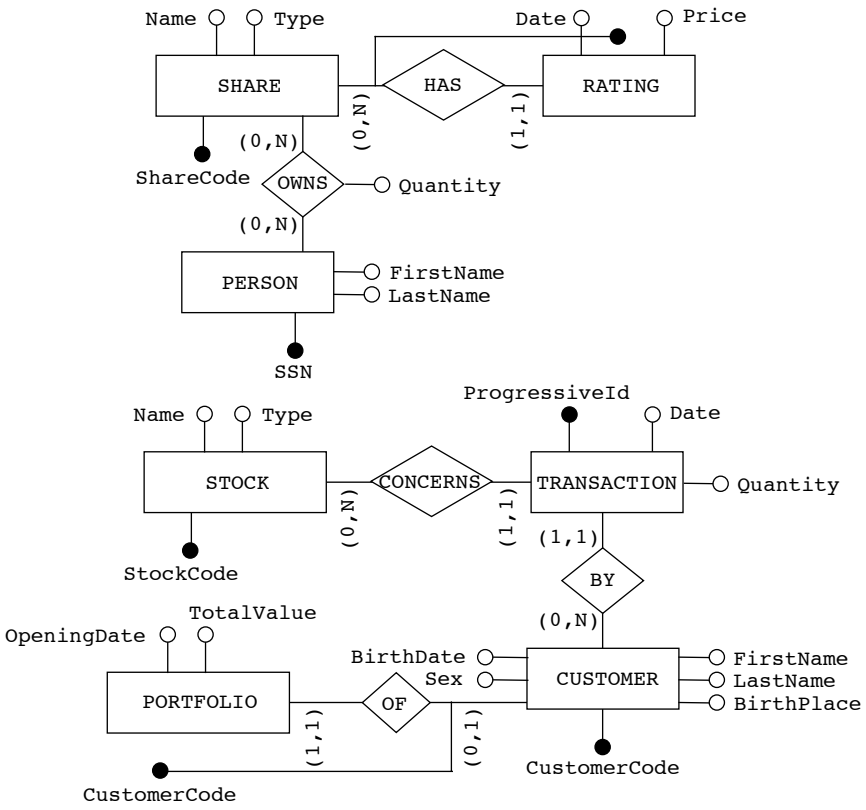
```
GATE_TYPES(TypeId,Description)
GATES(GateId,GateName,Highway,County:COUNTIES,TypeId:GATE_TYPES)
COUNTIES(County,State)
DISTANCES(FromGate:GATES,ToGate:GATES,DistanceInKm,RangeId:DISTANCE_RANGES)
DISTANCE_RANGES(RangeId,FromKms,ToKms,Charge)
TOLLS(TollId,(FromGate,ToGate):DISTANCES,TimeIn,DateIn,TimeOut,DateOut)
```

The reconciled database can now be easily obtained by merging corresponding concepts (we assumed that `DistanceRange` and the `FromKms-ToKms` attributes carry the same information).

```
GATE_TYPES(TypeId,Description)
GATES(GateId,GateName,Highway,County:COUNTIES,TypeId:GATE_TYPES)
COUNTIES(County,State)
DISTANCES(FromGate:GATES,ToGate:GATES,DistanceInKm,RangeId:DISTANCE_RANGES)
DISTANCE_RANGES(RangeId,FromKms,ToKms,Charge)
TOLLS(TollId,(FromGate,ToGate):DISTANCES,TimeIn,DateIn,TimeOut,DateOut)
```

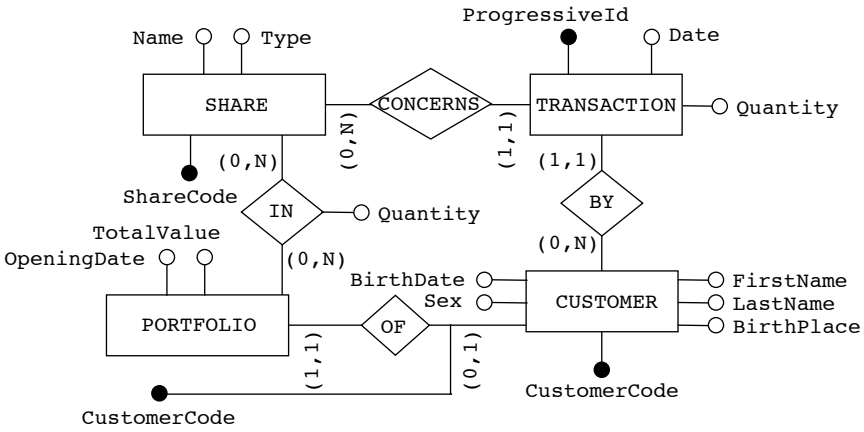
✔ Exercise 1.4

The two Entity-Relationship schemata resulting from inspection and normalization are shown below.



To obtain the reconciled schema, you simply have to:

- solve the stock-share and customer-person synonymies;
- recognize that the ownership relationship between shares and customers is better modeled through the portfolio concept;
- merge the corresponding concepts.


ProgressiveId
Name
Type
Date
SHARE
CONCERNS
TRANSACTION
Quantity
(0,N)
(0,N)
(1,1)
(1,1)
ShareCode
IN
Quantity
BY
TotalValue
OpeningDate
(0,N)
(0,N)
BirthDate
Sex
CUSTOMER
FirstName
LastName
BirthPlace
PORTFOLIO
OF
(1,1)
(0,1)
CustomerCode
CustomerCode

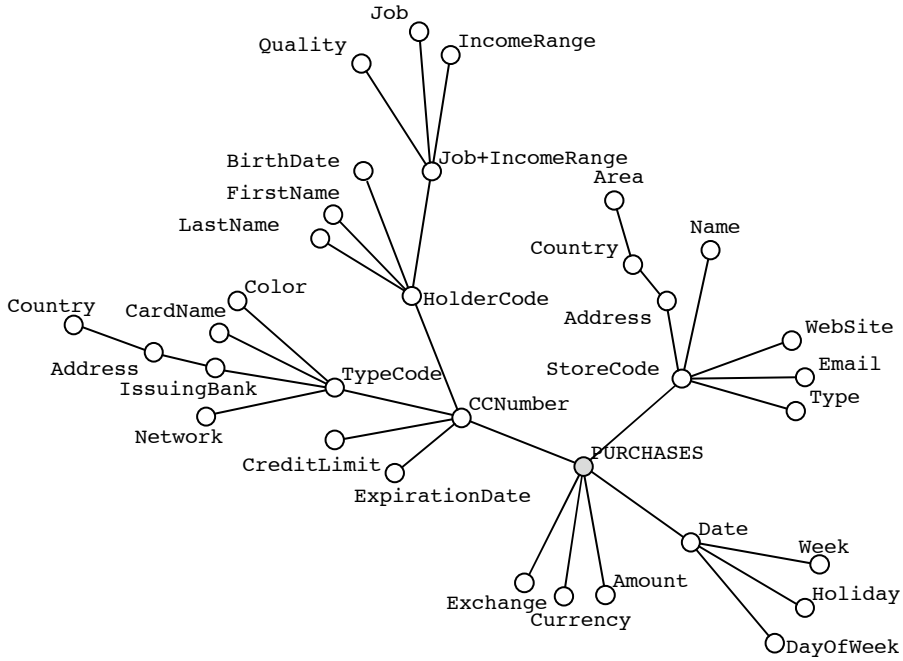
Part 2:
CONCEPTUAL DESIGN

✔ Exercise 2.1

Adding the missing functional dependencies leads to the following normalized schema:

```
PURCHASES(PurchCode,Date,Amount,Currency,Exchange,CCNumber:CREDIT_CARDS,
     StoreCode:STORES)
CREDIT_CARDS(CCNumber,TypeCode:CARD_TYPES,HolderCode:HOLDERS,ExpirationDate,
     CreditLimit)
HOLDERS(HolderCode,FirstName,LastName,BirthDate,(Job,IncomeRange):QUALITIES)
QUALITIES(Job,IncomeRange,Quality)
CARD_TYPES(TypeCode,Network,IssuingBank:BANKS,CardName,Color)
BANKS(IssuingBank,Address:ADDRESSES)
ADDRESSES(Address,Country:COUNTRIES)
COUNTRIES(Country,Area)
STORES(StoreCode,Name,Address:ADDRESSES,WebSite,Email,Type)
```

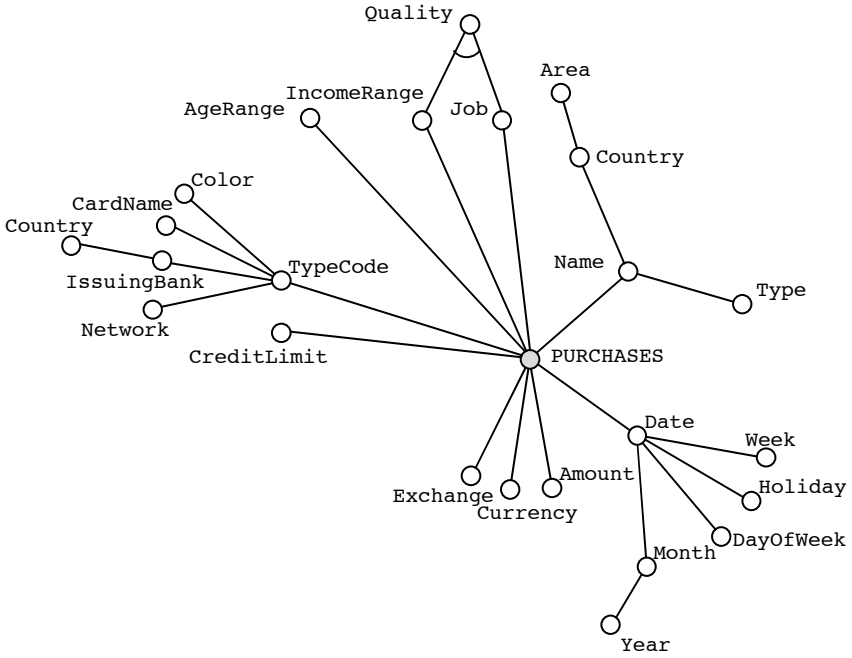
On this schema, the only relation that can be chosen as fact is PURCHASES. The attribute tree with root in PURCHASES is depicted below. Note that relation QUALITIES could not be reached by the tree-building algorithm.



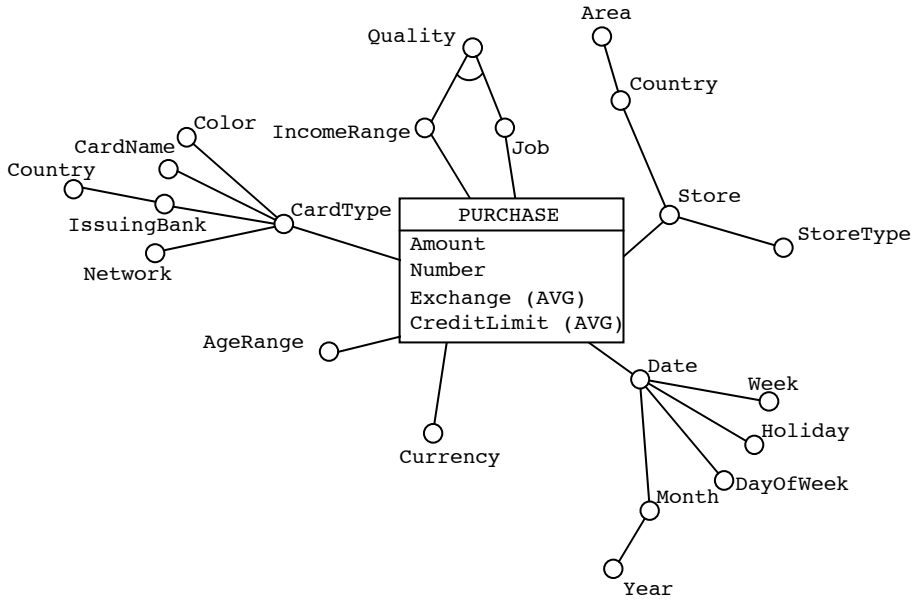
Some reasonable editing on this attribute tree leads to the reduced attribute tree shown below, where:

- a time hierarchy has been added;
- the store web-site and email have been pruned;
- Job+IncomeRange has been grafted and it has been recognized that Quality is a cross-dimensional attribute;
- Name has been assumed to be a candidate key for STORES, so it has been swapped with StoreCode and StoreCode has been pruned;
- the store and the bank addresses have been grafted;
- CCNumber has been grafted, which means losing the credit card granularity and pruning ExpirationDate;

- `BirthDate` has been replaced by a less-detailed `AgeRange` attribute;
- `HolderCode` has been grafted, together with his `FirstName` and `LastName`.

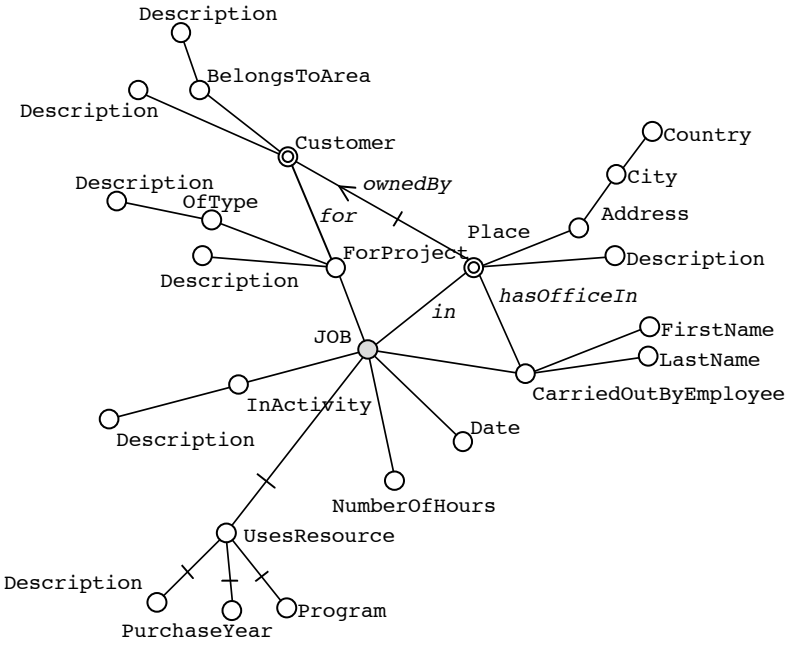


`Exchange`, `CreditLimit`, and `Amount` are chosen as measures, while all the other root children are chosen as dimensions. The resulting fact schema is depicted below. Note that measures `Exchange` and `CreditLimit` are non-additive.

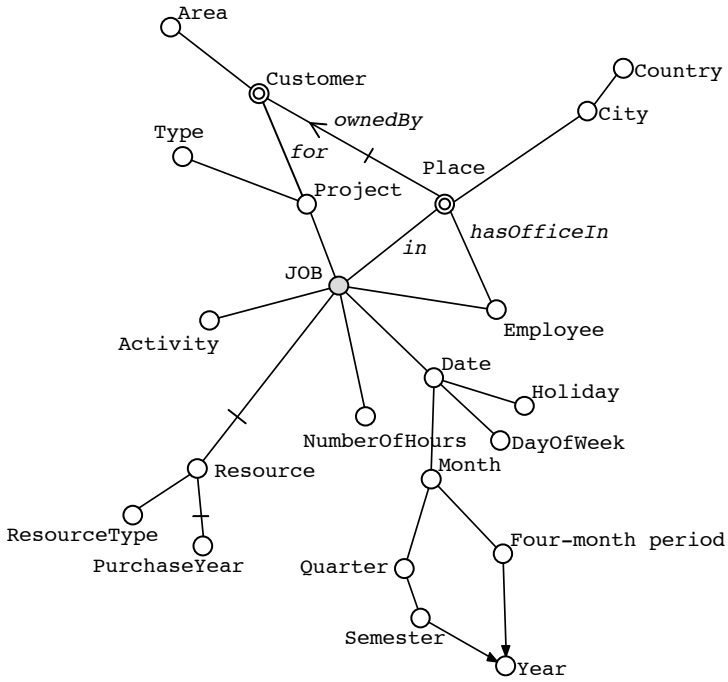


✔ Exercise 2.2

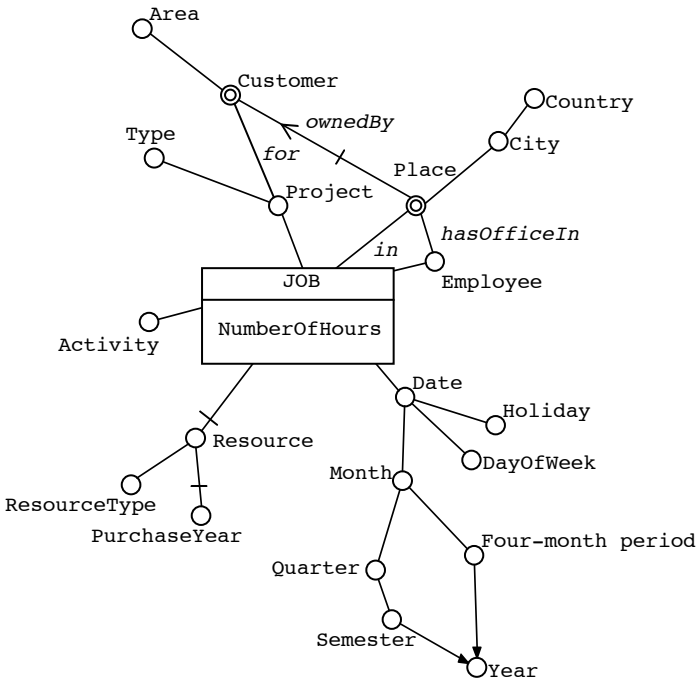
The fact for this source schema is entity `JOB`; the related attribute tree is shown below. The direction of the arc from `Place` to `Customer` is explicitly shown to avoid ambiguity.



In editing the attribute tree, all description attributes are assumed to be candidate keys for their entity, so they are swapped with the corresponding identifiers. In particular, the `FirstName-LastName` pair is assumed to be a unique identifier for employees, so it is swapped with `EmployeeId`. Besides, a `ResourceType` flag is added as a child of `Resource` to distinguish hardware from software resources.

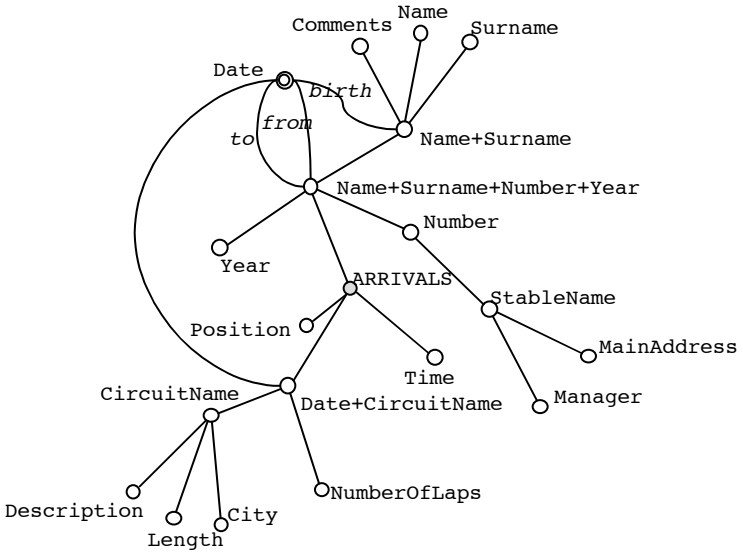


After choosing dimensions and measures, the fact schema below is obtained.

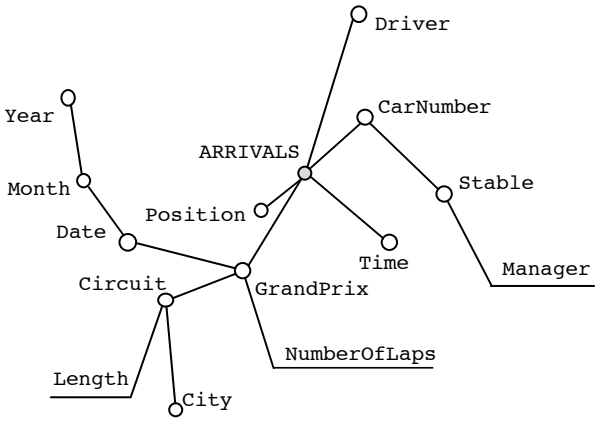


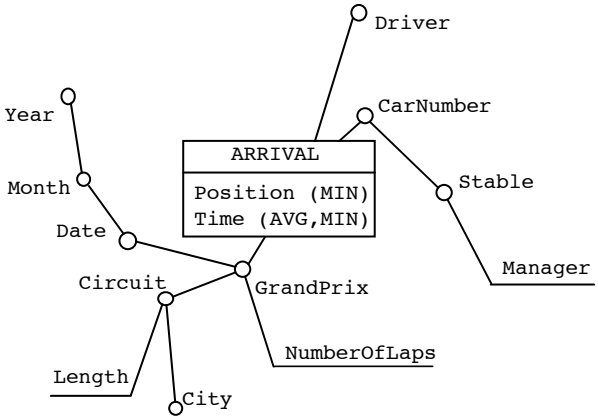
✔ Exercise 2.3

Choosing `ARRIVALS` as a fact allows for the largest possible attribute tree to be built.



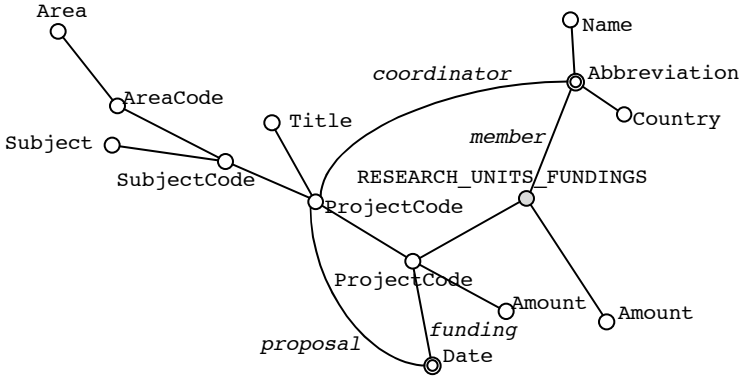
When editing the tree, the `Year` attribute (a child of `Name+Surname+Number+Year`) is pruned because it always has the same value of the other `Year` attribute (a descendant of `Date+CircuitName`).



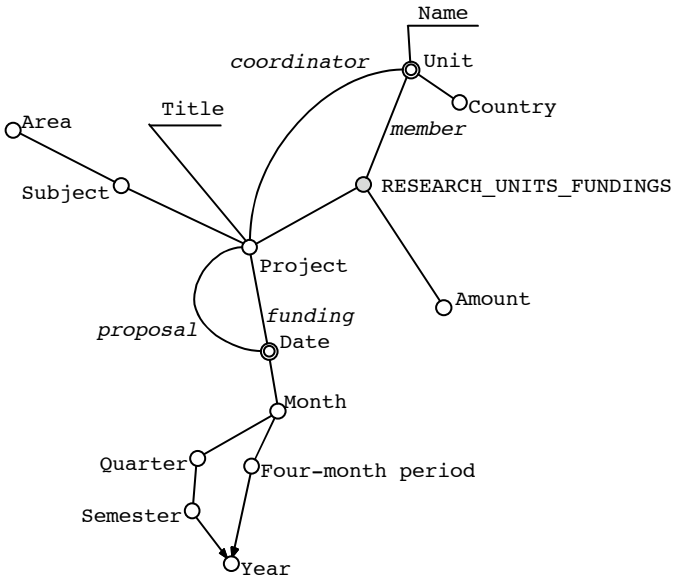


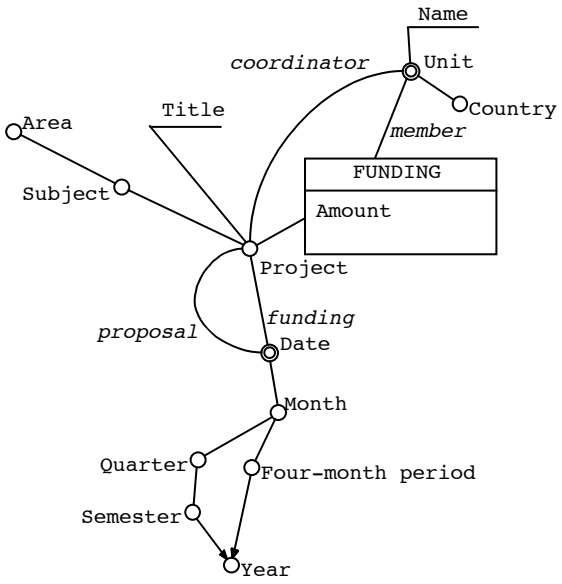
✔ Exercise 2.4

Choosing `RESEARCH_UNITS_FUNDINGS` as a fact leads to the attribute tree below.



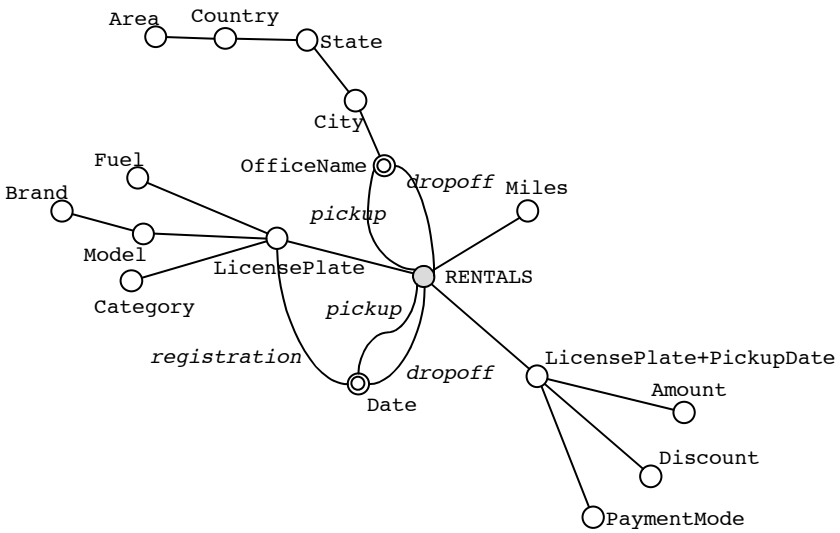
In the edited attribute tree, the amount of projects is pruned because it can be calculated by aggregating the amounts of the single project units. Besides, the one-to-one arc between `ProjectCode` and `ProjectCode` is eliminated by grafting its second end.



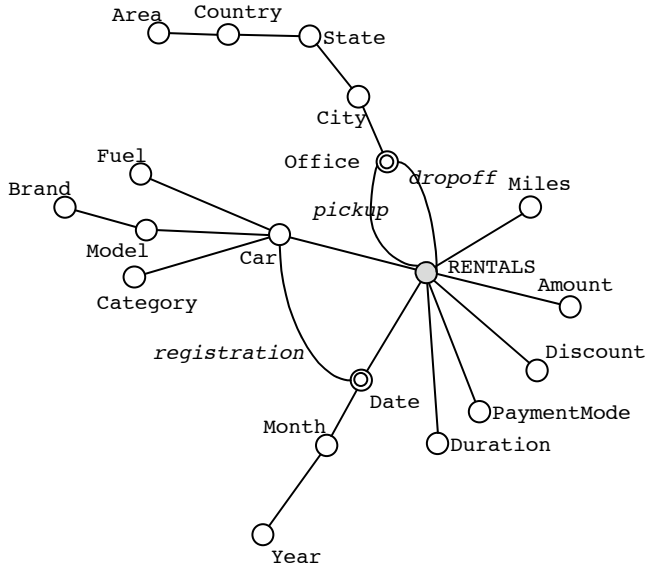


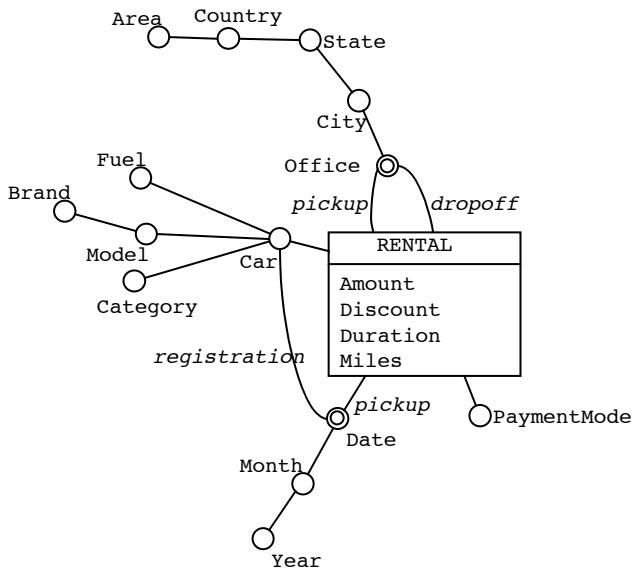
✔ Exercise 2.5

Choosing either RENTALS or PAYMENTS as facts is the same here, because these two relations are related by a one-to-one association.



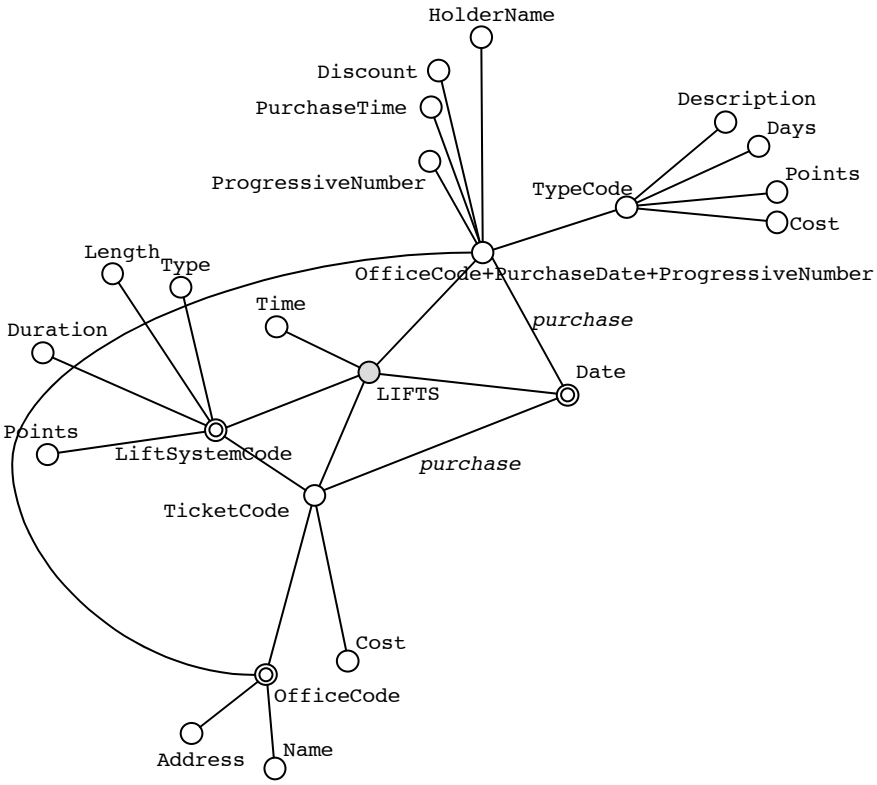
In the edited attribute tree, the drop-off date is pruned and replaced by a `Duration` attribute computed as the number of days between the drop-off and the pick-up dates.





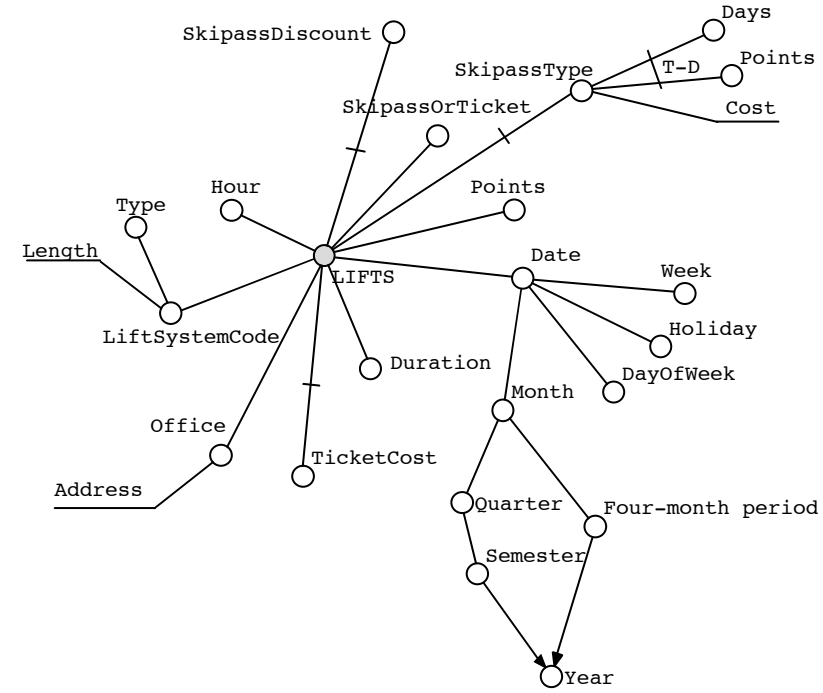
✔ Exercise 2.6

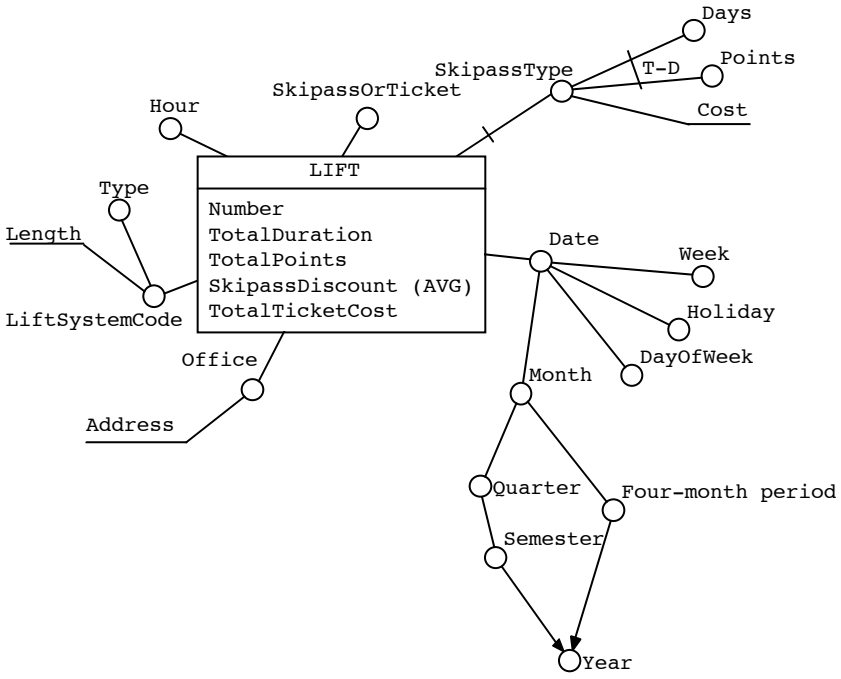
The attribute tree with root in `LIFTS` is depicted below.



Note that, in the edited tree, the functional dependency from `LiftSystemCode` to `Points` has been removed to make `Points` a child of the root, so that it can be chosen as a measure. The same is done for `Duration`. The ticket and the skipass granularities are removed, and a `SkipassOrTicket` flag is added to

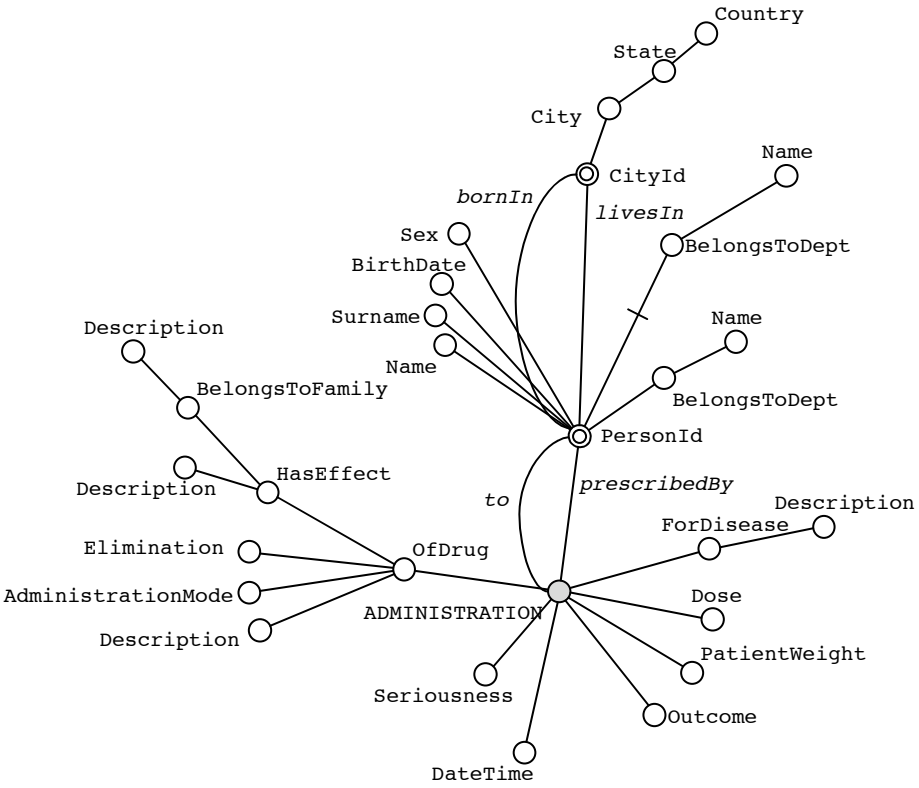
show whether each lift was paid with a skipass or a ticket. Finally, `Time` (in HH:MM:SS format) is transformed in `Hour`.



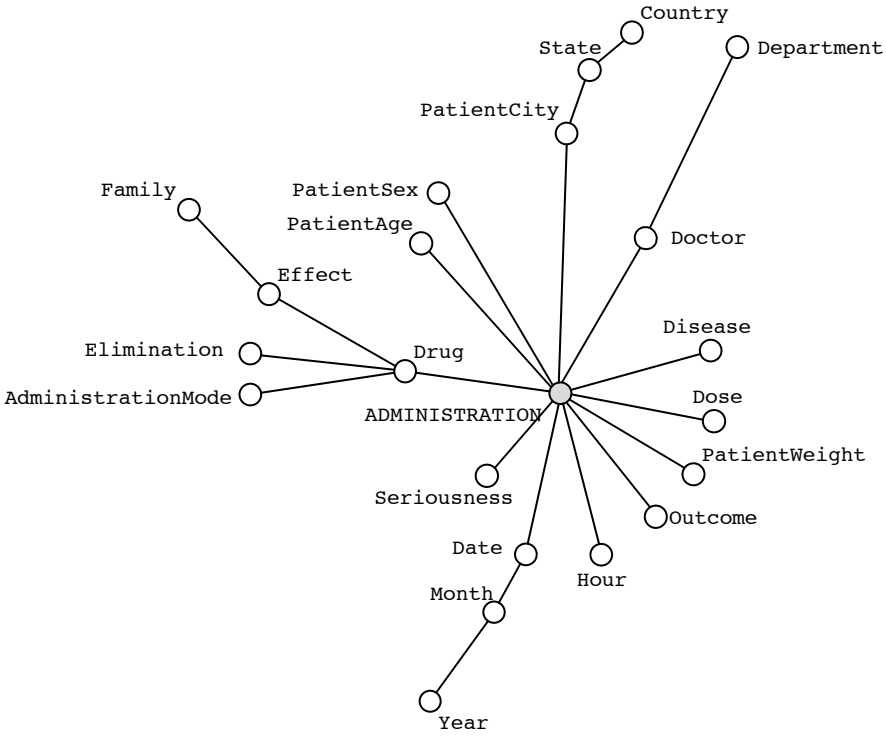


✔ Exercise 2.7

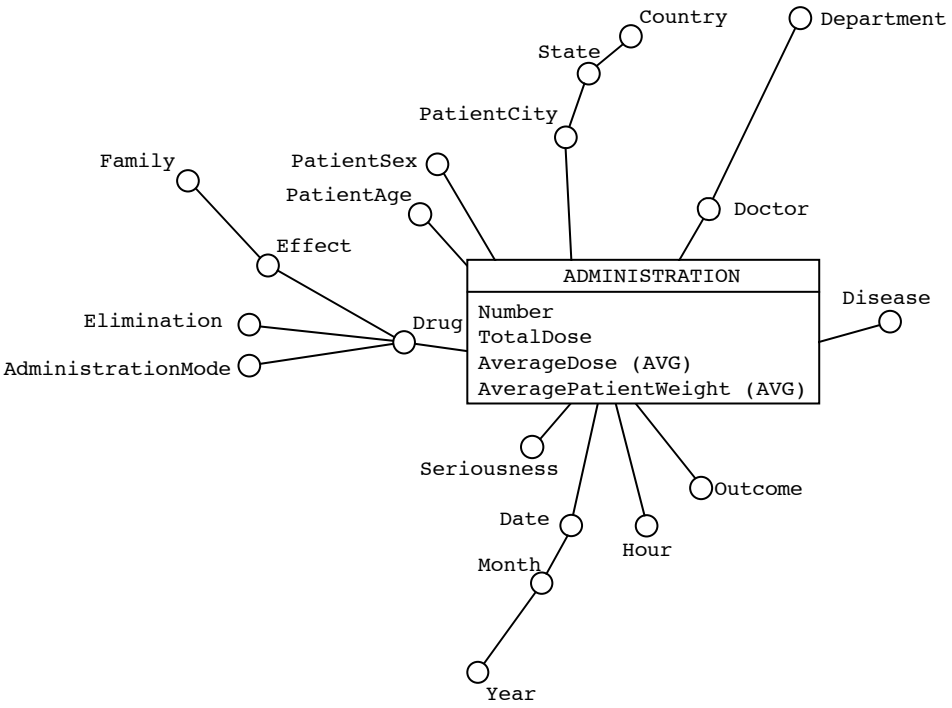
The attribute tree with root in `AMINISTRATION` is shown below.



In the edited tree, the `DateTime` attribute is split into a `Date` and a `Hour` attributes. All description attributes are assumed to be candidate identifiers. As concerns the shared hierarchies on `PersonId`, the patient granularity is removed while the doctor granularity is retained.



The fact schema obtained by choosing `PatientWeight` and `Dose` as measures is shown below.


Country
Department
State
PatientCity
Family
PatientSex
PatientAge
Doctor
Effect
ADMINISTRATION
Disease
Elimination
Drug
Number
TotalDose
AverageDose (AVG)
AveragePatientWeight (AVG)
AdministrationMode
Seriousness
Outcome
Date
Hour
Month
Year

Part 3:
LOGICAL DESIGN

✔ Exercise 3.1

A star schema for PURCHASE is reported below. Note that IncomeRange and Job have been treated as degenerate dimensions and, like AgeRange and Currency, incorporated in the fact table to save on joins.

```
DT_CARD_TYPE(idCardType,CardType,Network,IssuingBank,Country,CardName,Color)
DT_STORE(idStore,Store,Country,Area,StoreType)
DT_DATE(idDate,Date,Week,Holiday,DayOfWeek,Month,Year)
BT_QUALITY(IncomeRange,Job,Quality)
FT_PURCHASE(idCardType:DT_CARD_TYPE,idStore:DT_STORE,idDate:DT_DATE,
  IncomeRange,Job,AgeRange,Currency,Amount,Number,Exchange,CreditLimit)
```

✔ Exercise 3.2

A snowflake schema for JOB is reported below.

```
DT_PROJECT(idProject,Project,Type,forCustomer:DT_CUSTOMER)
DT_EMPLOYEE(idEmployee,Employee,hasOfficeInPlace:DT_PLACE)
DT_PLACE(idPlace,Place,ownedByCustomer:DT_CUSTOMER,City,Country)
DT_CUSTOMER(idCustomer,Customer,Area)
DT_DATE(idDate,Date,DayOfWeek,Holiday,Month,Quarter,Four-MonthPeriod,Semester,
  Year)
DT_RESOURCE(idResource,Resource,ResourceType,PurchaseYear)
FT_JOB(idProject:DT_PROJECT,inPlace:DT_PLACE,idDate:DT_DATE,
  idEmployee:DT_EMPLOYEE,idResource:DT_RESOURCE,Activity,NumberOfHours)
```

✔ Exercise 3.3

A star schema for ARRIVAL is reported below. In this case we choose to store drivers into a dimension table.

```
DT_GRAND_PRIX(idGrandPrix,GrandPrix,NumberOfLaps,Circuit,Length,City,Date,Month,
  Year)
DT_CAR(idCarNumber,CarNumber,Stable,Manager)
DT_DRIVER(idDriver,Driver)
FT_ARRIVAL(idGrandPrix:DT_GRAND_PRIX,idCarNumber:DT_CAR,idDriver:DT_DRIVER,
  NumberOfHours)
```

✔ Exercise 3.4

A snowflake schema for FUNDING is reported below.

```
DT_UNIT(idUnit,Unit,Name,Country)
DT_PROJECT(idProject,Project,coordinatorUnit:DT_UNIT,Title,Subject,Area,
  fundingDate:DT_DATE,proposalDate:DT_DATE)
DT_DATE(idDate,Date,Month,Quarter,Four-MonthPeriod,Semester,Year)
FT_FUNDING(idProject:DT_PROJECT,memberUnit:DT_UNIT,Amount)
```

✔ Exercise 3.5

A snowflake schema for RENTAL is reported below.

```
DT_CAR(idCar,Car,Fuel,Model,Brand,Category,registrationDate:DT_DATE)
DT_OFFICE(idOffice,Office,City,State,Country,Area)
DT_DATE(idDate,Date,Month,Year)
FT_RENTAL(pickupOffice:DT_OFFICE,dropoffOffice:DT_OFFICE,idCar:DT_CAR,
  pickupDate:DT_DATE,PaymentMode,Amount,Discount,Duration,Miles)
```

✔ Exercise 3.6

A star schema for LIFT is reported below. Considering that both degenerate dimensions have low-cardinality domains, in this case we choose to use a junk table for them.

```
DT_SKIPASS(idSkipassType,SkipassType,Cost,Points,Days)
DT_LIFT_SYSTEM(idLiftSystemCode,LiftSystemCode,Length,Type)
DT_OFFICE(idOffice,Office,Address)
DT_DATE(idDate,Date,DayOfWeek,Week,Holiday,Month,Quarter,Four-MonthPeriod,
  Semester,Year)
JT(idJunk,SkipassOrTicket,Hour)
FT_LIFT(idLiftSystemCode:DT_LIFT_SYSTEM,idOffice:DT_OFFICE,idDate:DT_DATE,
  idSkipassType:DT_SKIPASS,idJunk:JT,Number,TotalDuration,TotalPoints,
  SkipassDiscount,TotalTicketCost)
```

✔ Exercise 3.7

A star schema for ADMINISTRATION is reported below. We use a junk table for degenerate dimensions with low-cardinality domains.

```
DT_DRUG(idDrug,Drug,Effect,Family,Elimination,AdministrationMode)
DT_CITY(idCity,PatientCity,State,Country)
DT_DOCTOR(idDoctor,Doctor,Department)
DT_DATE(idDate,Date,Month,Year)
JT(idJunk,Hour,Seriousness,PatientSex,Outcome)
FT_ADMINISTRATION(idDrug:DT_DRUG,idCity:DT_CITY,idDate:DT_DATE,
  idDoctor:DT_DOCTOR,idJunk:JT,PatientAge,Number,TotalDose,AverageDose,
  AveragePatientWeight)
```

✔ Exercise 3.8

If degenerate dimensions are stored in separate dimension tables, the total space used can be computed as follows:

```
FACT_TABLE(idS:SEX_DT,idA:AGE_DT,idC:COUNTRY_DT,...)
```
100 000 tuples × 4 bytes × 3 surrogate keys = 1 200 000 bytes
```
SEX_DT(idS,sex)
```
2 tuples × (4 + 1) bytes = 10 bytes
```
AGE_DT(idA,age)
```
5 tuples × (4 + 5) bytes = 45 bytes
```
COUNTRY_DT(idC,country)
```

30 tuples × (4 + 20) bytes = 720 bytes

Total space = 1 200 775 bytes

If degenerate dimensions are stored in the fact table (without using surrogates), the total space is:

```
FACT_TABLE(sex,age,country,...)
```
100 000 tuples × (1 + 5 + 20) bytes = 2 600 000 bytes

Finally, if a junk table is created to store all three dimensions, the total space is:

```
FACT_TABLE(idJ:JUNK_TABLE,...)
```
100 000 tuples × 4 bytes = 400 000 bytes
```
JUNK_TABLE(idJ,sex,age,country)
```
(2 × 5 × 30) tuples × (4 + 1 + 5 + 20) bytes = 9000 bytes

Total space = 409 000 bytes

The most economic solution turns out to be the one using a junk table.

✔ Exercise 3.9

```
DT_A(idA,a,b,c,idE:DT_E)
```
100 000 tuples × (4 + 20 + 20 + 20 + 4) bytes = 6 800 000 bytes
```
DT_E(idE,e,h)
```
100 tuples × (4 + 20 + 20) bytes = 4400 bytes
```
BRIDGE(idD:DT_D,c,weight)
```
3 × 10 000 tuples × (4 + 20 + 4) bytes = 840 000 bytes
```
DT_D(idD,d,idE:DT_E,f,g)
```
1000 tuples × (4 + 20 + 4 + 20 + 20) bytes = 68 000 bytes

Total space = 7 712 400 bytes

✔ Exercise 3.10

The first solution consists in using a bridge table to model the association between groups of diagnoses and diagnoses:

```
DT_PATIENTS(idP,patient,age)
```
10 000 tuples × (4 + 20 + 20) bytes = 440 000 bytes
```
DT_DATES(idD,date)
```
1095 tuples × (4 + 20) bytes = 26 280 bytes
```
DT_WARDS(idW,ward,department)
```
20 tuples × (4 + 20 + 20) bytes = 880 bytes
```
DT_DIAGNOSES(idDiag,diagnosis,category)
```
100 tuples × (4 + 20 + 20) bytes = 4400 bytes
```
BRIDGE(idGroup,idDiag:DT_DIAGNOSIS,weight)
```
(54 750 × 3) tuples × (4 + 4 + 4) bytes = 1 971 000 bytes

FT(idP:DT_PATIENTS,idD:DT_DATES,idW:DT_WARDS,idGroup,cost,duration,numberOfTests)
(50 × 365 × 3) tuples × (4 + 4 + 4 + 4 + 20 + 20 + 20) bytes = 4 161 000 bytes

Total space = 6 603 560 bytes

Note that attribute idGroup is not a foreign key because there is no relation where it is primary key. Nevertheless, the schema enables a natural join between the two idGroup attributes in FT and BRIDGE. In the second solution, the multiple arc is pushed down the fact:

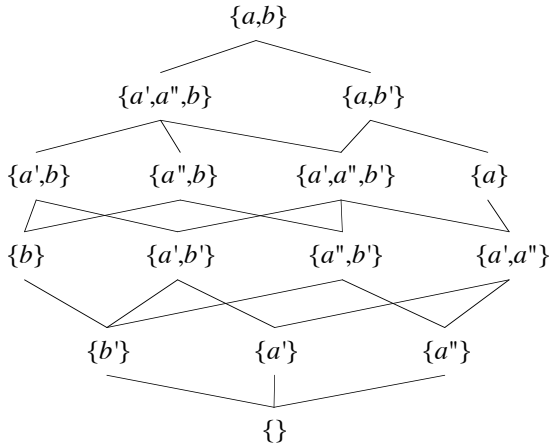
DT_PATIENTS(idP,patient,age)
10 000 tuples × (4 + 20 + 20) bytes = 440 000 bytes
DT_DATES(idD,date)
1095 tuples × (4 + 20) bytes = 26 280 bytes
DT_WARDS(idW,ward,department)
20 tuples × (4 + 20 + 20) bytes = 880 bytes
DT_DIAGNOSES(idDiag,diagnosis,category)
100 tuples × (4 + 20 + 20) bytes = 4400 bytes
FT(idP:DT_PATIENTS,idD:DT_DATES,idW:DT_WARDS,idDiag:DT_DIAGNOSES,
idGroup,cost,duration,numberOfTests)
(50 × 365 × 3 × 3) tuples × (4 × 5 + 20 + 20 + 20) bytes = 13 140 000 bytes

Total space = 12 954 560 bytes

The second solution is clearly overtaken by the first one.

✔ Exercise 3.11

The multidimensional lattice is depicted below. Note that the presence of a branch on the *a* hierarchy gives rise to the *a*', *a*" combination in the group-by sets.



✔ Exercise 3.12

In the first solution, the bridge table DT_C that models the cross-dimensional attribute *c* references attribute *b*:

DT_A(idA,a,b)
1000 tuples × (4 + 100 + 100) bytes = 204 000 bytes
DT_D(idD,d)
50 tuples × (4 + 100) bytes = 5200 bytes
DT_C(b,idD:DT_D,c,e)
(500 × 50) tuples × (100 + 4 + 100 + 100) bytes = 7 600 000 bytes
---
Total space = 7 809 200 bytes

In the second solution, redundant surrogates of *b* are introduced in DT_A to reduce the space taken by the bridge table:

DT_A(idA,a,b,idB)
1000 tuples × (4 + 100 + 100 + 4) bytes = 208 000 bytes
DT_D(idD,d)
50 tuples × (4 + 100) bytes = 5200 bytes
DT_C(idB,idD:DT_D,c,e)
(500 × 50) tuples × (4 + 4 + 100 + 100) bytes = 5 200 000 bytes
---
Total space = 5 413 200 bytes

Finally, in the third solution a snowflaking on *b* allows for space to be further reduced:

DT_A(idA,a,idB:DT_B)
1000 tuples × (4 + 100 + 4) bytes = 108 000 bytes
DT_B(idB,b)
500 tuples × (4 + 100) bytes = 52 000 bytes
DT_D(idD,d)
50 tuples × (4 + 100) bytes = 5200 bytes
DT_C(idB:DT_B,idD:DT_D,c,e)
(500 × 50) tuples × (4 + 4 + 100 + 100) bytes = 5 200 000 bytes
---
Total space = 5 365 200 bytes

✔ Exercise 3.13

In the first solution, *a* is directly included in both the fact and the bridge tables:

BRIDGE(a,idGroup,weight)
(3 × 1 000 000) tuples × (100 + 4 + 4) bytes = 324 000 000 bytes
FT(a,idGroup,...)
1 000 000 tuples × (100 + 4 + 300) bytes = 404 000 000 bytes

Total space = 728 000 000 bytes

In the second solution, a surrogate is introduced to reduce the storage space:

`DT_A(idA,a)`	
1000 tuples × (4 + 100) bytes =	104 000 bytes
`BRIDGE(idA:DT_A,idGroup,weight)`	
(3 × 1 000 000) tuples × (4 + 4 + 4) bytes =	36 000 000 bytes
`FT(idA:DT_A,idGroup,...)`	
1 000 000 tuples × (4 + 4 + 300) bytes =	308 000 000 bytes

Total space =	344 104 000 bytes