Databases, exercises

Legend: [●] detailed solution, [○] final result, [–] no solution

1. Exercises

1.1. ER modeling

1. The monthly meals of a students' restaurant shall be stored in a database. Each day the meal contains a soup, a main course and a dessert. One dish can occur multiple times in the same month, but we know that to each soup – main course combination, only one dessert is suitable. For each dish, we want to store its name, its number of calories, and the list of ingredients with the amount which is needed from the given ingredient for the given dish. Create an ER diagram for this database [–]

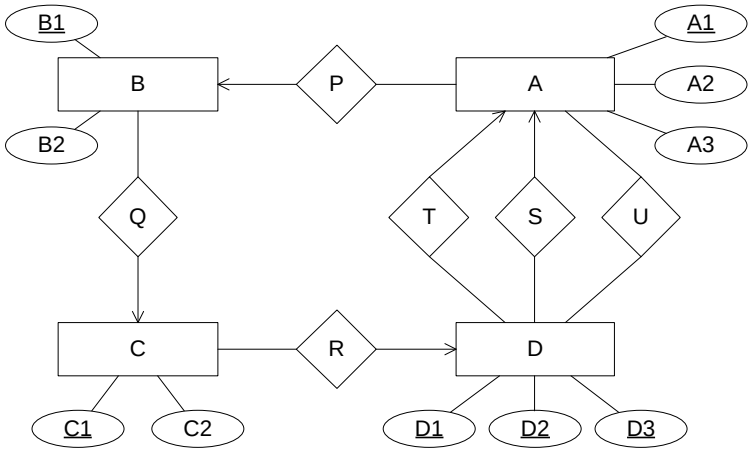
2. Given is the following, rough description:
   A patient might have multiple diseases, there are diseases which nobody has at the moment. Each patient is treated at a single facility, by possible multiple doctors. A doctor might have multiple patients, and these patients might be located at different facilities. A facility might be empty, and belongs to a single hospital. One doctor is employed by a maximum of 3 hospitals. A hospital is always lead by a director, who is a doctor of the given hospital, has an economy degree, and does not work in other hospitals. Create an ER diagram about the above! The diagram should represent the functionality (cardinality) of relationship sets. Identify entities with suitably chosen attributes, define the keys! [●]

3. Given is the following, rough description:
   A hospital consists of several departments, each of which has a department leader chief doctor and an arbitrary number of chief doctors. If there is no department leader chief doctor then there is a commissioned department leader (who might not be a chief doctor). All of these people are employees of the hospital, possess a medical degree and are not employed by any other hospital. Besides them, the hospital has several further employees: doctors, nurses, support staff. Doctors and nurses always work at a specific department, while support staff can directly belong to the hospital. Each employee has their employee ID, but for doctors, their Doctors' Association ID numbers are stored as well. The hospital is always led by a doctor of the given hospital, who has an economy degree too, and is not employed in any other hospital. A patient, once they get into the hospital, might be treated in multiple departments until cured, and in the meantime, might be treated with different diseases.
   Create an ER diagram about the above. Using the learned syntax, include the cardinality of each relationship set. Identify the entities with suitably chosen attributes, and underline the keys. [●]

4. How can an ER diagram containing a ternary relationship set be transformed to an equivalent ER diagram that only contains binary relationship sets? [●]

1.2. Relational schemas, relational algebra

5. Transform the below ER diagram into relational schemes. Use as few relational schemes as possible! [–]



6. In this exercise, work on the ER diagrams you created in exercises 2 and 3.
   a) Transform the diagrams to relational schemas!
   b) Perform the transformation so that you define as few as possible schemas for representing relationship types. [●]

7. Given are two relational schemes *R*(*A*, *B*) and *S*(*B*, *C*) furthermore two relations *r*(*R*) and *s*(*S*). We know that the union of *r* and *s* includes (*f*, *c*) and (*d*, *e*). We also know that their natural join includes the following elements:

A	*B*	*C*
a	b	a
f	c	g

   Determine the two relations! [–]

8. Given are relations *r* and *s*, and their schemes *R*(*A*, *B*) and *S*(*B*, *C*) respectively. *r* has n_r different rows, while *s* has n_s different rows. What is the maximal and minimal number of rows in the natural join of the two relations (as a function of n_r and n_s), if
   - *A* is a key in *R*,
   - *B* is a key in *R*,
   - *B* is a key in both *R* and *S*,
   - *A* is a key in *R*, and *B* is a key in *S*. [●]

9. If the cardinality of attribute *A* is less than the number of elements of the domain of *A*, then *A* cannot be a (simple) key. Prove this statement right or wrong. [–]

10. Let *r* and *s* be two relations with the same attributes. Let furthermore *X* be a subset of this common attribute set. Which of the following statements are true?

a) $\pi_X(r \cup s) = \pi_X(r) \cup \pi_X(s)$
b) $\pi_X(r \backslash s) = \pi_X(r) \backslash \pi_X(s)$ [–]

11. Given the following scheme description, provide relational algebra expressions for the questions below.
PRODUCT(*MANUFACTURER*, *MODEL*, *TYPE*)
PC(*MODEL*, *SPEED*, *RAM*, *HDD*, *CD*, *PRICE*)
LAPTOP(*MODEL*, *SPEED*, *RAM*, *HDD*, *SCREEN*, *PRICE*)
PRINTER(*MODEL*, *COLOR*, *TYPE*, *PRICE*)
Questions:
    - Which PC models have a speed of at least 1500?
    - Which manufacturers produce a laptop with an at least 1000 GB HDD?
    - Provide the model number and price of each product made by manufacturer B, regardless of their type.
    - Which manufacturers produce laptops but not PCs?
    - Which manufacturers produce at least two different PCs or laptops with a speed of at least 3 GHz? (There are no two identical model numbers!)
    - All PCs faster than 3 GHz, and their manufacturers
    - Those manufacturers, who produce laptops that have the same parameters as a PC made by themselves (and possibly other laptops too)
    - Those manufacturers, who produce PCs that have the same parameters as a laptop made by them (and possibly other PCs too) [○]

12. Given is the below database scheme describing a star fleet:
STARSHIP(SHIPNAME, YEAR, SPECIES)
WORKER(WORKERNAME, ID, BIRTH)
POSITION(ID, SHIPNAME, RANK)
The meaning of relational schemes is the following:
Starship: name, and production year of a starship, and the name of the species that designed the starship
Worker: name, star fleet identifier, date of birth
Position: which worker works on which starship and in which position
Give a relational algebra expression listing the workers working on the ship of Captain Catherine Janeway. [–]

13. Given are the following relations (with the obvious semantics):
likes(*person*, *beer*), *sells*(*pub*, *beer*), *visits*(*person*, *pub*).
Using relational algebra, express
    a) the list of those beers which are liked by every person visiting the bars selling the beer.
    b) the list of those persons, who like each beer sold in all pubs visited by them. [○]

1.3. Physical organization

28. In what ways can multiple key search be supported? [–]

29. A file of 25 000 records shall be stored using a sparse index. Record length is 850 bytes, block capacity (excluding the header) is 4000 bytes. Key size is 50 bytes, pointer size is 18 bytes.
    - At least how many blocks are needed for storing the entire structure?
    - How much time does it take at most to read the contents of a record, if the RAM has 6000 bytes free? (One block operation takes 5 milliseconds.)
    - If we have 10 times as much free RAM, can the record access time be reduced? What if we have 100 times as much RAM? How should we use this extra RAM? [–]

30. A file of 15 525 records shall be stored using a sparse index. Record length is 850 bytes, block capacity (excluding the header) is 4000 bytes. Key size is 50 bytes, pointer size is 18 bytes.
    - At least how many blocks are needed for storing the entire structure?
    - How much time does it take at most to read the contents of a record, if the RAM has 6000 bytes free? (One block operation takes 5 milliseconds.)
    - If we have 10 times as much free RAM, can the record access time be reduced? What if we have 100 times as much RAM? How should we use this extra RAM? [–]

31. A file shall be stored using a dense index and a sparse index built on top of the dense index. Give a reasonable estimation for the number of necessary blocks, given the below conditions:
    - the file contains $3 \cdot 10^6$ records,
    - one record is 300 bytes,
    - one block is 1000 bytes,
    - key size is 45 bytes,
    - pointer size is 5 bytes. [○]

32. A file of 270 000 records shall be stored. We can choose from two possibilities: building a single-layer sparse index on top a dense index, or we build a three-layered sparse index. Which solution can be realized with fewer blocks if we furthermore want to achieve that 20% of each and every block in the main file and the index files is unused (i.e. no block can be filled more than 80% of its capacity)? One block is 1900 bytes, one record is 300 bytes, the key size is 35 bytes, and the size of a pointer is 15 bytes. (We assume optimal storage that is, beyond fulfilling the above requirements our data uses the fewest possible blocks) [○]

33. One billion records shall be stored in a database. Record size is 100 bytes, block size is 4000 bytes, one block operation takes 5 milliseconds. We have two keys, both 10 bytes in size. Pointers occupy 32 bits. For simplicity we assume that a single block fits in the RAM at a time. We also assume that records are stored in the most compact way.
    - Suggest a storage method if we would like to search with both keys. Search can take 40 milliseconds at maximum. The method shall support interval searches too. Create an explanatory figure about your structure.

- A given search is expected to return 8% of all records. Suggest the most efficient search method you can think of. [●]

34. Using bucket hashing, what should be modified on the storage structure to reduce data access time to its half? [–]

35. We are designing a hash based data structure for storing data on CDs in our database. For each CD we store whether the given CD stores pictures, music, videos or data. We ase a single character field for this the corresponding values of which are P, M, V and D, respectively. What kind of hash function should we use if we'd like to base our hash on this field? What is the cardinality of the field? [–]

36. 1 000 000 records shall be stored in a database using bucket hashing. The size of a record is 110 bytes, one block is 3000 bytes, a key is 25 bytes, and a pointer is 64 bytes long. Block access time is 5 ms. Record access time can be 20 ms at most. The hash table fits in the RAM, and the hash function spreads the records evenly.
    - What is the average record access time?
    - How many bytes does the hash table occupy from the RAM?
    - How much extra RAM would be needed to reduce record access time to its half? [●]

1.4. Functional dependencies

37. Prove that Armstrong's axioms can be deduced from the below three rules.
    If *X, Y, Z, C* are attribute sets of a relational schema, then:
    B1. $X \to X$
    B2. if $X \to YZ$ and $Z \to C$ then $X \to YZC$
    B3. if $X \to YZ$ then $X \to Y$. [–]

38. Show that the transitivity axiom is true! [–]

39. Prove the expandability axiom! [–]

40. Is the below set of axioms complete that is, are all logical consequences deducible from them?
    - If $X \subseteq R$ then $X \to X$.
    - If $X, Y \subseteq R$ and $X \to Y$, then $XW \to YW$ is true for an arbitrary $W \subseteq R$.
    - If $X, Y, Z \subseteq R$, $X \to Y$ and $Y \to Z$, then $X \to Z$. [–]

41. Provide a relation *r* matching schema *R(A, B, C)*, where *r* has 4 rows, and no non-trivial functional dependency is true on it. [●]

43. Are the below rules true? (A, B, C, D are arbitrary sets of attributes on schema R.)
    a) $A \to B, C \to D \Rightarrow A \cup (C \setminus B) \to BD$,
    b) $A \to B, C \to D \Rightarrow C \cup (D \setminus A) \to BD$. [○]

44. Relation r matches schema $R(A,B,C)$, and has 3 rows. Prove that there exists a nontrivial functional dependency that *r* fulfills! [●]

1.5. Normal forms

45. Show that a 2NF relation can be redundant. Explain how its redundancy can be eliminated. Give an example for a 2NF non-redundant relation having at least 3 elements. [–]

46. Prove that dependency sets *F* and *G* are equivalent exactly if $F \subseteq G^+$ and $G \subseteq F^+$.[–]

47. What is the highest normal form of schema $R(A,B,C,D)$, if $F = \{C \to B, B \to D, AB \to AC, CD \to B\}$? [○]

48. What is the highest normal form of $R(I,S,T,Q)$ if its dependency set is $F = \{I \to Q, ST \to Q, IS \to T, QS \to I\}$ [○]

49. Prova that if *R* is not BCNF, then $\exists A,B$, where $A,B \in R$ and $R \setminus AB \to A$ [●]

50. Determine whether a relation matchning (*R, F*) can contain redundancy and if yes, what kind of redundancy. *R*(*X, Y, Z, W*), $F = \{XY \to Z, YZ \to W, X \to W, WY \to X\}$. [–]

1.6. Transaction management

62. Consider the following scheduling of transactions T_1, T_2, T_3, and T_4 (consecutive operations are listed from left to right):
    T_2: RLOCK *A*; T_3: RLOCK *A*; T_2: WLOCK *B*; T_2: UNLOCK *A*;
    T_3: WLOCK *A*; T_2: UNLOCK *B*; T_1: RLOCK *B*; T_3: UNLOCK *A*;
    T_4: RLOCK *B*; T_1: RLOCK *A*; T_4: UNLOCK *B*; T_1: WLOCK *C*;
    T_1: UNLOCK *A*; T_4: WLOCK *A*; T_4: UNLOCK *A*; T_1: UNLOCK *B*;
    T_1: UNLOCK *C*.
    Draw the precedence graph of the scheduling, and decide whether the scheduling is serializable! [○]

63. Draw the precedence graph. Use locks. How does the graph change if the system is two-phase? [●]

T_1	T_2
	WRITE *A*
WRITE *B*	
	WRITE *B*
WRITE *A*	

64. What happens according to the redo protocol if the transaction aborts in the indicated steps (1-6)? [–]

	log(*T*, BEGIN)
1.	
	LOCK(*A*)
	LOCK(*B*)
2.	
	log(*T*, <old value of *A*>, <new value of *A*>)
	log(*T*, <old value of *B*>, <new value of *B*>)
3.	
	log(*T*, COMMIT)
4.	
	WRITE(*A*)
	WRITE(*B*)
5.	
	UNLOCK(*A*)
	UNLOCK(*B*)
6.	

65. Is the following transaction strict 2PL? If not, modify it to make it strict 2PL. What does this protocol guarantee? [○]

LOCK *A*
READ *A*
$A = A \times 2$
WRITE *A*
COMMIT
UNLOCK *A*

66. Why can it be beneficial to use locks with timestamp-based transaction management? [–]

67. *Optional exercise:* Is the below scheduling serializable with timestamp-based (R/W) scheduling? [●]

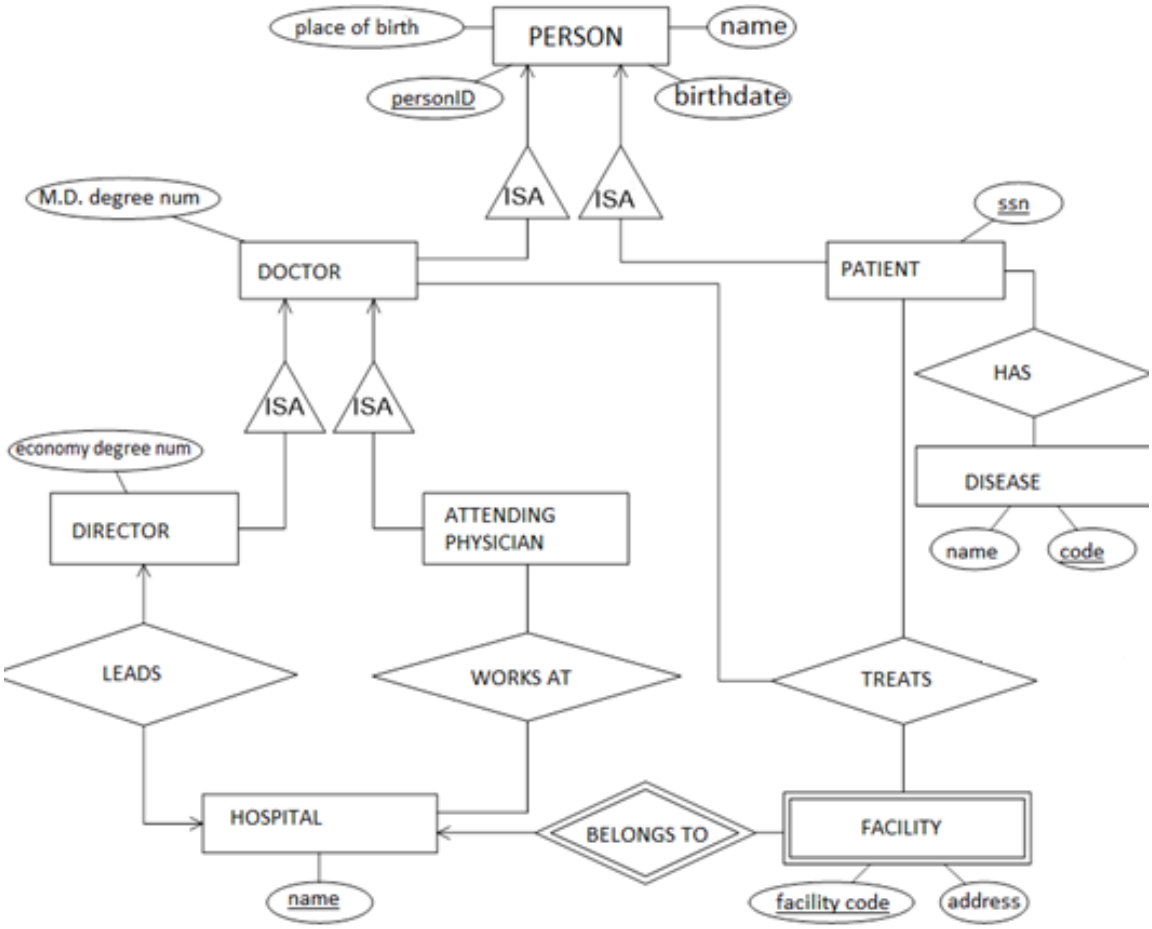
	T_1 $t(T_1) = 10$	T_2 $t(T_2) = 20$
(1)	READ *A*	
(2)		WRITE *A*
(3)	WRITE *A*	

2. Solutions of exercises

2.1. ER modeling

Exercise 2

The ER diagram:

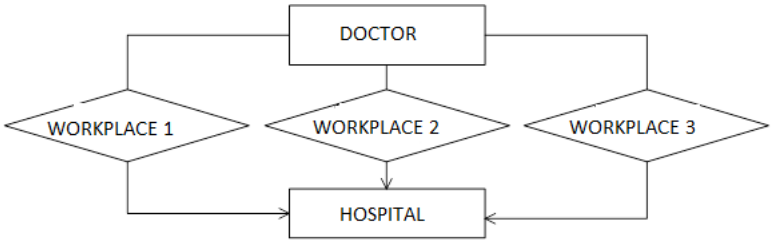


Notes::

- The exercise didn't specify whether the identifiers of facilities are unique globally or only within the same hospital. Thus, we could make Facility a weak entity set, which is beneficial as this way we can guarantee that each facility belongs to *exactly* one hospital.
- The solution cannot model that the director is also an employee of the given hospital. One possible solution for this: Instead of making Director and Attending Physicial a specialized entity set of Doctor, we transform Works At to a weak entity set between Doctor and Hospital, and create weak entity set Director as a specialized entity set of Works At. Weak entity set Director will furthermore have

attribute „economy degree num”. The drawback of this solution is that it doesn’t constrain the uniqueness of Director.

- We have not defined cardinality for ternary relationship sets. Thus, Treats cannot model that a patient is treated in a single facility.
- The diagram cannot model that a doctor is employed by at most 3 hospitals, so this has to be included as a side note.[1] Another possible solution is defining 3 many-to-one relationships:

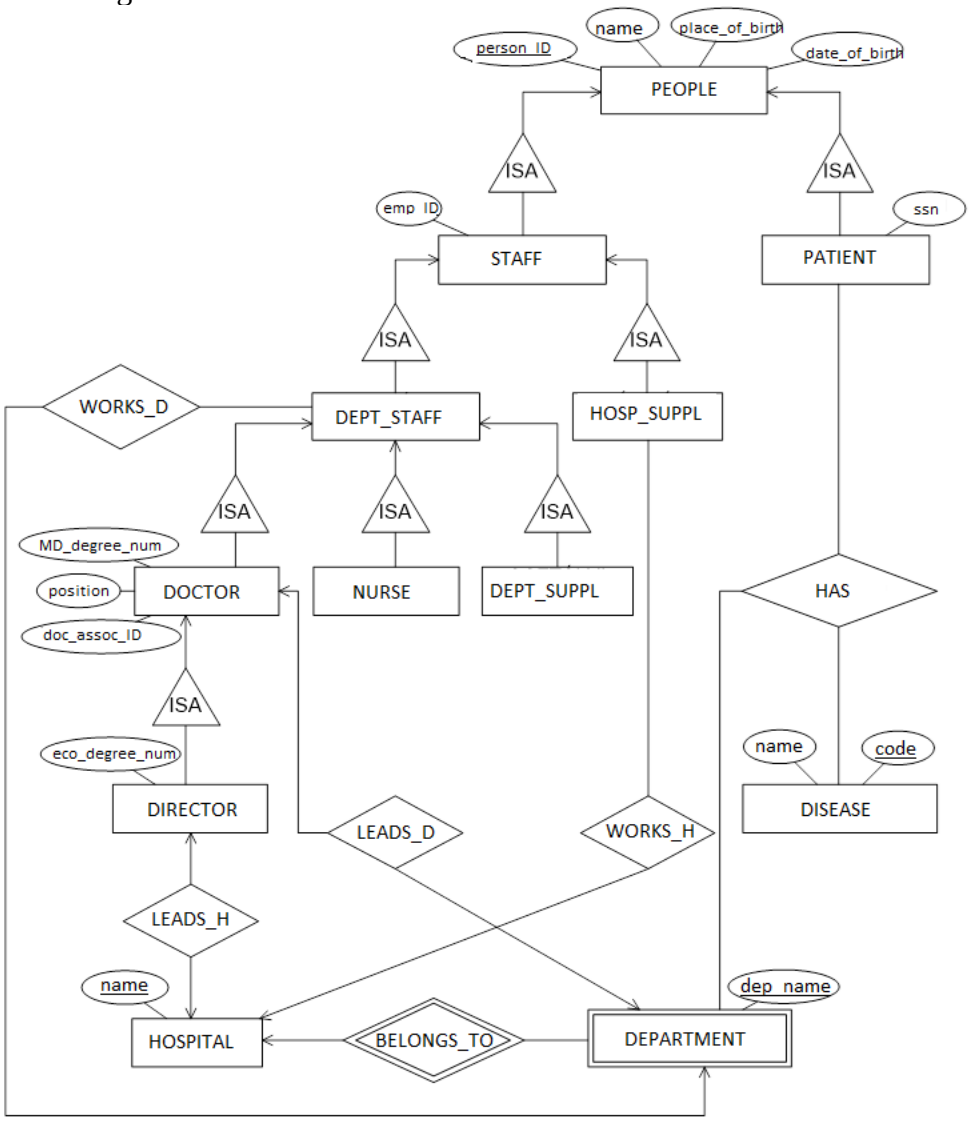


This method can be used well if only a few relationship sets have to be defined. The advantage of this method is that many-to-one relationship sets can be mapped to a fewer relational schemes, as can be seen in the solution of Exercise 6.

[1] In a database using relational data model, these restrictions can be guaranteed by using so-called consistency conditions (check constraints) or triggers.

Exercise 3

The ER diagram:



Meaning of abbreviated relationship sets:

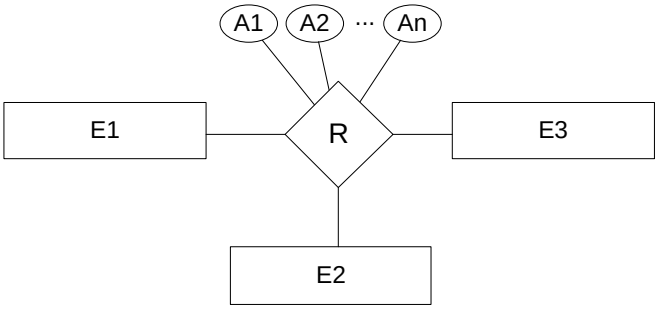
- WORKS_D: works at department,
- WORKS_H: works at hospital,
- LEADS_D: leads department,
- LEADS_H: leads hospital.

Department leader chief doctors and commissioned department leaders are identified by the *position* attribute.

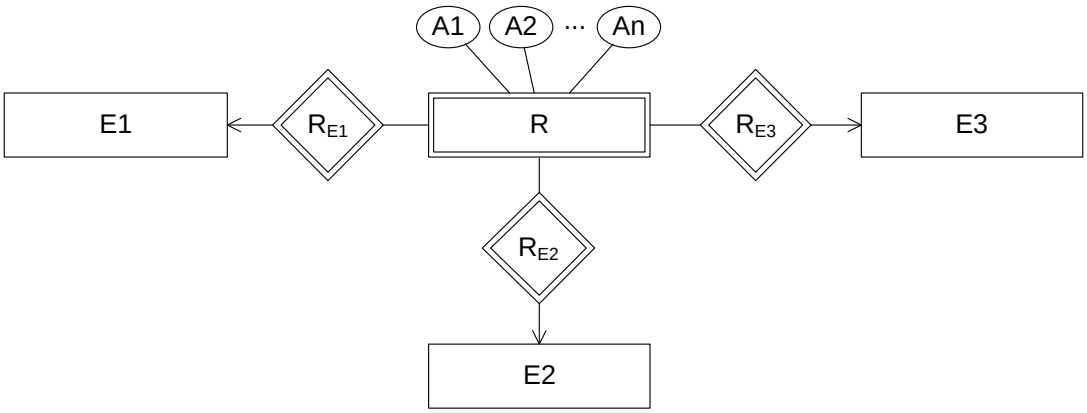
Similarly to the solution of Exercise 2, this solution can neither model that the director is an employee of the directed hospital. A workaround similar to that of Exercise 2 can be applied here as well.

Exercise 4

Example for ternary relationship set:



The ternary relationship set is transformed to three binary relationship sets. For this, we create an entity set R, representing the relationship set. The attributes of the relationship set generally do not provide uniqueness, so R will be a weak entity set the elements of which are identified by the keys of entity sets E1, E2, and E3.



Analyzing the cardinality of ternary relationship types go beyond the objective of the course and thus, is omitted here.[2]

2.2. Relational schemas, relational algebra

Exercise 6

For schemas the keys or foreign keys of which are not clear from the notation of the schema, we included the attributes of the keys separately. In the case of schemas where each attribute in itself is a foreign key, all attributes together form the key.
If in part *b)* the attributes of a schema are not listed then they are the same as in part *a)*.

[2] For those interested in deeper insights, check out for example Trevor H. Jones, Il-Yeol Song, *Binary Equivalents of Ternary Relationships in Entity-Relationship Modeling: a Logical Decomposition Approach*, Journal of Database Management, April–June, 2000, pp. 12–19, http://www.ischool.drexel.edu/faculty/song/publications/p_JDB99.PDF

Exercise 2

a) Schemas corresponding to entity sets and one-to-one relationship sets:

- PERSON(personID, place_of_birth, birthdate, name)
- DOCTOR(personID, place_of_birth, birthdate, name, MD_degree_num)
    - DOCTOR(personID, MD_degree_num), key: person_id, alternative solution
- PATIENT(personID, place_of_birth, birthdate, name, ssn)
    - PATIENT(personID, ssn), key: person_id, alternative solution
- DIRECTOR(personID, place_of_birth, birthdate, name, MD_degree_num, economy_degree_num, leads_hospital_name)
    - DIRECTOR(personID, MD_degree_num, economy_degree_num, leads_hospital_name), key: person_id, alternative solution
- ATTENDING_PHYSICIAN(personID, place_of_birth, birthdate, name, MD_degree_num)
    - ATTENDING_PHYSICIAN(personID, MD_degree_num), key: person_id, alternative solution
- HOSPITAL(name)
- DISEASE(code, name)
- FACILITY(facility_code, address, hospital_name), key: hospital_name, facility_code

Schemas corresponding to binary many-to-many and ternary relationship type

- HAS(personID, disease_code)
- TREATS(doctor_personID, patient_personID, hospital_name, facility_code), key: doctor_personID, patient_personID, hospital_name, facility_code
- WORKS_AT(person_id, hospital_name)

The above alternative solutions work like this: For each specialized entity, a relation element is created in the relation of the parent schema. This stores the common attribute values for the specialized entity. Furthermore, a relation element is created for the specielized entity too, in the corresponding relation. This will store the attribute values corresponding to the specialized entity. The key of the latter will be also a foreign key for the former. This way, the key constraint will hold in the mapped relational schema as well (i.e. for example it will not be possible that a director and a patient have the same personID).

Note 4 of solution of Exercise 2 – the constraint of one doctor being employed at most three hospitals cannot be represented on an ER diagram – holds after the transformation as well. If we created three many-to-one relationship types as suggested in the above note, then these can be mapped to the following schemas instead of a single WORKS_AT schema::

- WORKS_AT_1(personID, hospital_name)
- WORKS_AT_2(personID, hospital_name)
- WORKS_AT_3(personID, hospital_name)

b) Schemas corresponding to entity sets and one-to-one relationship sets: alternative mappings for specialized entity sets are not used, PERSON schema is eliminated (as all

relevant people are either doctors or patients[3]). The personID and economy degree number of the director will be stored in the HOSPITAL schema (guaranteeing that each hospital can have at most one director). Thus, the DIRECTOR schema will be unnecessary.

- PATIENT
- DOCTOR
- ATTENDING_PHYSICIAN
- HOSPITAL (name, director_personID, director_economy_degree_num)
- DISEASE
- FACILITY

Schemas of binary many-to-many and ternary relationship sets are unchanged

- HAS
- TREATS
- WORKS_AT

The 3 relationship sets recommended in note 4 of the solution of Exercise 2 can be mapped to the ATTENDING_PHYSICIAN scheme too, in which case the WORKS_AT_1 etc. schemas in part *a)* can be replaced by attributed in the ATTENDING_PHYSICIAN schema:

- ATTENDING_PHYSICIAN(personID, place_of_birth, birthdate, name, MD_degree_num, works_at_1, works_at_2, works_at_3)

Note: of course, in this case the WORKS_AT schema shall be elliminated.

Exercise 3

a) Schemas corresponding to entity sets and one-to-one relationship sets (alternative solutions for specialized entity sets work the same way as in Exercise 2 – these will not be listed here):

- PEOPLE(person_ID, name, place_of_birth, date_of_birth)
- STAFF(person_ID, name, place_of_birth, date_of_birth, emp_ID)
- PATIENT(person_ID, name, place_of_birth, date_of_birth, ssn)
- DEPT_STAFF(person_ID, name, place_of_birth, date_of_birth, emp_ID)
- HOSP_SUPPL(person_ID, name, place_of_birth, date_of_birth, emp_ID)
- DOCTOR(person_ID, name, place_of_birth, date_of_birth, emp_ID, MD_degree_num, position, doc_assoc_ID)
- NURSE(person_ID, name, place_of_birth, date_of_birth, emp_ID)
- DEPT_SUPPL(person_ID, name, place_of_birth, date_of_birth, emp_ID)
- DIRECTOR(person_ID, name, place_of_birth, date_of_birth, emp_ID, MD_degree_num, position, doc_assoc_ID, eco_degree_num, leads_h_hospital_name)
- DISEASE(code, name)
- HOSPITAL(name)

[3] The PEOPLE schema is can be considered 'abstract' with object oriented programming terminology. Note: It is not abstract if the alternative solution for the mapping of Exercise 2 is used, as in that case, it will contain entities.

- DEPARTMENT(dep_name, hospital_name, leads_d_person_ID), key: dep_name, hospital_name

Schemas for many-to-one relationship types::

- WORKS_D(person_ID, hospital_name, dep_name), key: person_ID
- WORKS_H(person_ID, hospital_name), key: person_ID

Schema corresponding to the ternary relationship set:

- HAS(person_ID, hospital_name, department_name, disease_code)

Note: DEPARTMENT is a weak entity set, thus determining relationship set BELONGS_TO cannot be mapped to a separate schema, as that way DEPARTMENT would not have a key. In schema WORKS_D, the foreign key referring to DEPARTMENT is a composite foreign key containing the entire key of DEPARTMENT (dep_name, hospital_name). Thus, in WORKS_D, dep_name and hospital_name are not separate foreign keys.

b) Schemas corresponding to entity sets and one-to-one relationship sets: similarly to the schemas in Exercise 2, schema PEOPLE can be eliminated, and DOCTOR, and DIRECTOR can be converted to a single schema. The STAFF schema can be similarly eliminated. For many-to-one relationship types WORKS_H and WORKS_D, no separate schemas are created. For the latter, we include the appropriate foregin keys in DEPT_STAFF, while for the former one, we include the appropriate foreign key in HOSP_SUPPL.

- PATIENT
- DEPT_STAFF(person_ID, name, place_of_birth, date_of_birth, emp_ID, works_d_hospital_name, works_d_dep_name)
- HOSP_SUPPL(person_ID, name, place_of_birth, date_of_birth, emp_ID, works_h_hospital_name)
- DOCTOR(person_ID, name, place_of_birth, date_of_birth, emp_ID, MD_degree_num, position, doc_assoc_ID, works_d_hospital_name, works_d_dep_name)
- NURSE(person_ID, name, place_of_birth, date_of_birth, emp_ID, works_d_hospital_name, works_d_dep_name)
- DEPT_SUPPL(person_ID, name, place_of_birth, date_of_birth, emp_ID, works_d_hospital_name, works_d_dep_name)
- DISEASE
- HOSPITAL(name, director_person_ID, director_eco_degree_num)
- DEPARTMENT

Schema corresponding to the ternary relationship set:

- HAS

Exercise 8

Natural join will have at least 0 rows – 0 if and only if there is no common value of attribute *B* in *r(R)* and *s(S)*.

A* is a key in *R

At most $n_r \cdot n_s$, if the *B* values in each row of *r* and *s* are equal.

B* is a key in *R
Let's create the join by finding the matching rows in *r* for each row of *s*. Since *B* is a key in *R*, *r*'s attribute values on *B* are unique. Thus, we can find at most one matching row in *r* to each row of *s* that is, the resulting relation will have at most n_s rows.

B* is a key in both *R* and *S
The join attribute is a key, so is unique in each row, in both relations. Thus, after the join the rows can make up at most as many pairs as rows the relation with the fewer rows has. This means that the resulting relation has at most $min\{n_r, n_s\}$ rows.

A* is a key in *R*, *B* is a key in *S
From the viewpoint of the number of rows in the resulting relation, it is irrelevant that A is a key. The solution thus, is the opposite of that of the case, where *B* is a key in *R*: The resulting relation will have at mostn_r rows.

Exercise 11

The relation including the required data is denoted by *r*:

Which PC models have a speed of at least 1500?

$$r = \pi_{MODEL}\left(\sigma_{SPEED \geq '1500'}(pc)\right)$$

Which manufacturers produce a laptop with an at least 1000 GB HDD?

$$r = \pi_{MANUFACTURER}(\sigma_{HDD \geq '1000'}(laptop) \rhd\lhd product)$$

Provide the model number and price of each product made by manufacturer B, regardless of their type.

$$r = \sigma_{MANUFACTURER='B'}\big(\pi_{MANUFACTURER,MODEL,PRICE}(PRODUCT \rhd\lhd pc) \cup \pi_{MANUFACTURER,MODEL,PRICE}(product \rhd\lhd laptop) \cup \pi_{MANUFACTURER,MODEL,PRICE}(product \rhd\lhd printer)\big)$$

Which manufacturers produce laptops but not PCs?

$$\pi_{MANUFACTURER}(product \rhd\lhd laptop) \setminus \pi_{MANUFACTURER}(product \rhd\lhd pc)$$

Which manufacturers produce at least two different PCs or laptops with a speed of at least 3 GHz? (There are no two identical model numbers!)

$$s = \sigma_{SPEED \geq '3000'}\left(\pi_{MODEL,SPEED}(pc) \cup \pi_{MODEL,SPEED}(laptop)\right) \rhd\lhd product$$

$$t = \pi_{MANUFACTURER,MODEL}(s)$$

$$r = \pi_{MANUFACTURER}\left(t \underset{MANUFACTURER1 = MANUFACTURER2 \wedge MODEL1 \neq MODEL2}{\rhd\lhd} t\right)$$

All PCs faster than 3 GHz, and their manufacturers

$$s = \sigma_{SPEED > '3000'}\left(\pi_{MODEL,SPEED}(pc) \cup \pi_{MODEL,SPEED}(laptop)\right) \rhd\lhd product$$

$$r = \pi_{MANUFACTURER}(s)$$

Those manufacturers, who produce laptops that have the same parameters as a PC made by themselves (and possibly other laptops too).

$$l = \pi_{MANUFACTURER,MODEL,SPEED,RAM,HDD}(laptop \rhd\!\!\!\!\!\!\leftrightarrows\!\!\!\!\!\!\lhd\ termék)$$

$$p = \pi_{MANUFACTURER,MODEL,SPEED,RAM,HDD}(pc \rhd\!\!\!\!\!\!\leftrightarrows\!\!\!\!\!\!\lhd\ termék)$$

$$r = \pi_{l.MANUF.}\left(l \underset{l.MANUF. = p.MANUF \wedge l.SPD = p.SPD \wedge l.RAM = s.RAM \wedge l.HDD = p.HDD}{\rhd\!\!\lhd} p\right)$$

Those manufacturers, who produce PCs, that have the same parameters as a laptop made by them (and possibly other PCs too).
The solution is identical to that of the previous one.

Exercise 13

a) $\pi_{beer}(sells) \backslash \pi_{beer}(\pi_{person,beer}(sells \rhd\!\!\lhd visits) \backslash likes)$
b) The solution is very similar to that of part *a)*.

2.3. Physical organization

Exercise 31

The data file consists of $br = 10^6$ blocks.
The dense index needs $1{,}5 \cdot 10^5$ blocks.
The sparse index built on top of this dense index will need 7500 blocks.
In total, $10^6 + 1{,}5 \cdot 10^5 + 7{,}5 \cdot 10^3 = 1157500$ blocks are needed.

Exercise 32

The useful block size is 1520 bytes, the data file contains 54 000 blocks.

Single layer sparse index built on dense index:
The dense index needs 9000 blocks, the sparse index needs 300 blocks.
This is $54\,000 + 9000 + 300 = 63\,300$ blocksin total.

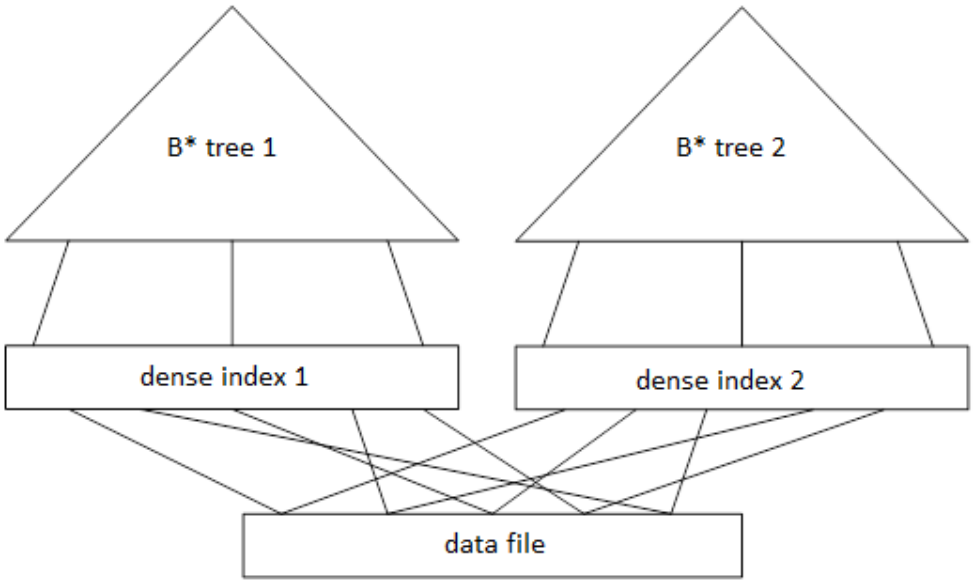
Three-layer sparse index
For storing the layers of the sparse index, 1800, 60, and 2 blocks are needed, respectively.
In total, this means $54000 + 1800 + 60 + 2 = 55862$ blocks, so this is the more economic solution.
Note: The three-layer sparse index is not a tree, as there are two blocks on the top layer.

Exercise 33

$n_r = 10^9$, $s_r = 100$ bytes, $b = 4000$ bytes, $k_1 = k_2 = 10$ bytes, $p = 32$ bits = 4 bytes, $t_{\text{block-op.}} = 5$ ms

One block fits $f_i = \left\lfloor \frac{b}{k+p} \right\rfloor = \left\lfloor \frac{4000}{10+4} \right\rfloor = 285$ index entries.

1. The need for interval searches excludes the possibility for hash organization. We need to use some kind of an index based organization. In order for a search to not need more than 40 ms, we can perform at most 8 block operations. One possible solution is to build dense indices for each search key, on top of the data file, and then we build a B* tree on top of each dense index.



Dense indices have 10^9 entries, for this $\left\lceil \frac{10^9}{285} \right\rceil = 3508772$ blocks are needed.

In case of a B* tree, we can fit an extra pointer into a block, even if its key does not have any space left. Here, we can do this, as $285 \cdot (10+4) + 4 = 3994$, so the branching factor of the tree will be 286. In order to be able to search in the dense index, we need a tree of $\lceil \log_{286} 3\,508\,772 \rceil = 3$ layers. Thus, to read a record, $3+1+1=5$ block operations are needed, which only takes 25 ms.

Note: In the course of solving this exercise, it has not yet been necessary to calculate the number of blocks in the data file due to the indirection provided by the dense index.

2. 8% of all records: $10^9 \cdot 0{,}08 = 8 \cdot 10^7$ records. Let's assume that the data file is *not ordered* by the search key (in case of two keys it is very unlikely to be sortable by both keys anyways). In this case, as reading one record takes 5 block operations, in order to read all the records included in the result, $(8 \cdot 10^7) \cdot 5 = 4 \cdot 10^8$ block operations are needed. If the block access time is 5 ms, this is more than 23 days.

The problem is caused by always having to traverse the entire index structure. Idea: if we only traverse the blocks of the dense index (3 508 772 block operations) and read the corresponding $8 \cdot 10^7$ records based on that, $(3\,508\,772 + 8 \cdot 10^7) \cdot 5\,\text{ms} \approx 4{,}8$ days are enough to retrieve all the needed records.

Interestingly, the best solution is traversing the data file itself. One block fits $f_r = \left\lfloor \frac{b}{s_r} \right\rfloor = \left\lfloor \frac{4000}{100} \right\rfloor = 40$ data records and thus, $b_r = \left\lceil \frac{n_r}{f_r} \right\rceil = \left\lceil \frac{10^9}{40} \right\rceil = 2{,}5 \cdot 10^7$ make up the data file. To traverse this, we "only" need $(2{,}5 \cdot 10^7) \cdot 5$ ms $\approx 1{,}5$ days.

Exercise 36

$n_r = 10^6$, $s_r = 110$ bytes, $b = 3000$ bytes, $k = 25$ bytes, $p = 64$ bits $= 8$ bytes, $t_{\text{record max.}} = 20$ ms, $t_{\text{block-op.}} = 5$ ms

1. According to the exercise, the hash table fits in the RAM and thus, to reach a data record we only need to read the bucket pointed to by the hash function, from beginning to end. Record access time can be 20 ms at maximum, one block access takes 5 ms, thus one bucket may contain up to 4 blocks. In best case, only the first block of the bucket has to be read (1 block operation), in worst case all of them (4 block operations). Thus, the average block access time is $t_{\text{average}} = \frac{1+4}{2} \cdot 5 \text{ ms} = 12{,}5 \text{ ms}$.

2. One data block fits $f_r = \left\lfloor \frac{b}{s_r} \right\rfloor = \left\lfloor \frac{3000}{110} \right\rfloor = 27$ records at maximum, one bucket contains $4 \cdot 27 = 108$ records in this case. The file contains $B = \left\lceil \frac{n_r}{108} \right\rceil = \left\lceil \frac{10^6}{108} \right\rceil = 9260$ buckets, for addressing which we need $9260 \cdot 8 = 74\,080$ bytes – and this is how much space the has table occupies in the RAM.

3. See Exercise 34. Here $\frac{N}{B} = \frac{10^6}{9260} \approx 108 >> 1$, thus the number of buckets have to be doubled, needing 74 080 bytes of extra RAM space – this is by how much the hash table will grow. Let's see how this affects record access times!

In case we have $B' = 9260 \cdot 2 = 18\,520$ buckets, $\left\lceil \frac{n_r}{B'} \right\rceil = \left\lceil \frac{10^6}{18\,520} \right\rceil = 54$ records will be in a single bucket, which needs $\left\lceil \frac{54}{f_r} \right\rceil = \left\lceil \frac{54}{27} \right\rceil = 2$ blocks per bucket.

Average record access time: $t_{\text{average}} = \frac{1+2}{2} \cdot 5 \text{ ms} = 7{,}5 \text{ ms}$.

Notes:

- We haven't considered that the last blocks of buckets are not full and thus, our average is not exactly precise. See the following example!

In case *a)* we store 5 records. To reach the first four, we need 1 block operation, while to reach the 5th, we need 2 block operations. This adds up to an average of $\frac{4\cdot 1+1\cdot 2}{5}=1{,}2$ block operations. In case *b)* 8 records are stored, so the average number of block operations is $\frac{4\cdot 1+4\cdot 2}{8}=1{,}5$. With the above method, we get $\frac{1+2}{2}=1{,}5$ block operations. Since this is an insignificant difference, but many calculations can be spared, we consider the block-level estimate to be good enough. Anyways, this is just an estimate, and the exact block access times depend on a number of factors.

- key size is irrelevant wrt the solution.

2.4. Functional dependencies

Exercise 37

The Armstrong axioms:

Reflexivity: If $Y \subseteq X$, then $X \to Y$
B1 can be trascribed to $X \to Y \cup (X \setminus Y)$, if $Y \subseteq X$. From this and B3: $X \to Y$.

Transitivity: If $X \to Y$ and $Y \to Z$, then $X \to Z$
Let us assume that $A \to B$ and $B \to D$ are true. Let's then use B2 with the below substitutions:

- $X = A$,
- $Y = \{\}$,
- $Z = B$,
- $C = D$.

We thus get $A \to BD$. Using B3, we get $A \to D$.

Expandability: If $X \to Y$, then $XZ \to YZ$
Let $A \to B$ true, and let F be an arbitrary attribute set. Due to B1, $AF \to AF$ is true. For this dependency and for $A \to B$, we then apply B2 with the below substitutions:

- $X = AF$,
- $Y = F$,
- $Z = A$,
- $C = B$.

As a result we get that $AF \to AFB$ is true. From here, using B3, we get $AF \to BF$.

Exercise 38

Transitivity axion: if $X \to Y$ and $Y \to Z$, then $X \to Z$.

According to the definition, $X \to Y$ holds if for each two rows $t, t' \in r(R)$ of the relation, in each point in time it is true that if $t[X] = t'[X]$, then $t[Y] = t'[Y]$ (where $t[Z]$ means $\pi_Z(t)$ that is, the projection of tuple *t* to attribute set *Z*.

1. Due to $X \to Y$ if $\exists t, t' \in r(R)$, that $t[X] = t'[X]$, then $t[Y] = t'[Y]$.
2. Due to $Y \to Z$ if $\exists u, u' \in r(R)$, that $u[Y] = u'[Y]$, then $u[Z] = u'[Z]$.

According to the first statement, if there are equal rows in *X*, then those are equal in *Y* too. According to the second statement, the rows that are equal in *Y*, are equal in *Z* too. By connecting the two statements, we can see that if (at any point in time) there are two rows that are equal on *X*, then these will be equal on *Z* too. This fulfills the requirement of dependency $X \to Z$.

Exercise 39

The expandability axiom says: If $X \to Y$, then $XZ \to YZ$.
Let us indirectly assume that $X \to Y$ holds, but $XZ \to YZ$ is not true. This means that if there exist rows t, and t' in a relation $r(R)$ so that $t[XZ] = t'[XZ]$, then $t[YZ] \neq t'[YZ]$. The attributes belonging to Z are obviously identical in rows t and t', as otherwise $t[XZ]$ and $t'[XZ]$ could not be identical. This means that $t[YZ]$ and $t'[YZ]$ differs in the value of attributes of *Y*, that is $t[Y] \neq t'[Y]$. But this is impossible as due to the original $X \to Y$ dependency, if $t[X] = t'[X]$, then $t[Y] = t'[Y]$. This is a contradiction, meaning that the original statement was true.

Exercise 40

False. Note that rule 2 is identical to the expandability axiom, and rule 3 is identical to the transitivity axiom.
The problem is that the reflexivity axiom (trivial dependency) cannot be deduced from the rules of the provided axioms. Thus, those $X \to Y$, dependencies where $Y \neq X$, but which are true due to $Y \subset X$ cannot be deducted from the provided axioms, if they have not been present in the original set of dependencies. For example, in case of the empty dependency set $F = \emptyset$:

- Rule 1 can only used to deducedependencies like $X \to X$.
- Using rule 2, we have to expand both sides of the dependencies at the same time. Since $Y \subset X$ and since we can only expand dependencies like $X \to X$, we cannot obtain dependencies like $X \to Y$.
- In order to get a dependency like $X \to Y$, we would need an attribute *Z* for which $X \to Z$ and $Z \to Y$. Such however, does not exist as using the other rules, we could not generate a pair of dependencies, where the right hand side of one dependency is the same as the left hand side of the other dependency.

Those $X \to Y$ dependencies, which are consequences of $Y \subset X$-ból thus, cannot necessarily be deduced (eg. in case of an empty dependency set), and thus this set of axioms is not complete.

Example: In case of relational schema $R(AB)$ and an empty dependency set, dependencies $AB \to A$, $AB \to B$, and trivial dependencies concerning the empty set ($A \to \varnothing$, $B \to \varnothing$, $AB \to \varnothing$) cannot be deduced.

- Using rule 1, trivial dependencies $A \to A$, $B \to B$, $AB \to AB$ and $\varnothing \to \varnothing$ can be deduced.
- Using rule 2, both sides of dependencies have to be expanded at the same time and thus, for example $AB \to A$ cannot be deduced.
- Rule 3 cannot be applied, as we do not have dependencies, where the left hand side of one dependency is identical to the right hand side of the other dependency.

Exercise 41

Let us examone what non-trivial dependencies can occur in case of attributes A, B, C.

The left hand side of dependencies can contain 3 attributes at maximum, but in case of 0 and 3 attributes, only trivial dependencies exist: $\varnothing \to \varnothing$, and $ABC \to \ldots$, where the right hand side contains an arbitrary (possibly empty) subset of ABC. Thus, we have to invastigate the cases where on the left side, there are 1 or 2 attributes.

One attribute on the left hand side:

- $A \to B, A \to C$
- $B \to A, B \to C$
- $C \to A, C \to B$

To break these dependencies, there must exist a pair of rows which are equal on one attribute but are different on other attributes.

A	B	C
0	1	1
1	0	1
1	1	0

The above example includes rows which are equal on *A*, on *B*, and on *C*, which break the above dependencies.

Two attributes on the left hand side

- $AB \to C$
- $AC \to B$
- $BC \to A$

In the above example, these dependencies are not violated by any pair of rows. In order to violate these 3 dependencies, we have to introduce rows so that there are pairs of rows which are equal of the two left hand side attributes of the dependency, but are not equal on its right hand side:

A	B	C
0	1	1
1	0	1
1	1	0
1	1	1

For this relation $r(R)$, no functional dependency holds.

Exercise 43

a) true

b) false

Exercise 44

When solving Exercise 41, we prove that in case of relational schema $R(A,B,C)$ the below dependencies can hold.

One attribute on the left hand side

- $A \to B$, $A \to C$
- $B \to A$, $B \to C$
- $C \to A$, $C \to B$

Two attributes on the left hand side

- $AB \to C$
- $AC \to B$
- $BC \to A$

Let's try to exclude the dependencies having two attributes on the left hand side. This can be done by inserting pairs of rows $t, t' \in r(R)$, which:

- are equal on *AB*, but not on *C*,
- are equal on *AC*, but not on *B*,
- are equal on *BC*, but not on *A*.

For this, we need at least two rows equal on *AB*, but in inequal on *C*. After this, we need a pair of rows, equal on *AC*, but not on *B*. Since the previous two rows were inequal on C, we need to find a new pair of rows. We can have at most 3 rows in total, so we can only introduce a single row – which has to be equal on *AC* with one of the previous rows. This however means that all three rows are equal on attribute *A*. Thus, we cannot introduce a pair of rows that break dependency $BC \to A$.

A	*B*	*C*
1	1	0
1	1	1
1	0	1

A 3-row relation *r* cannot violate all possible functional dependencies thus, we can always include a non-trivial dependency that holds on the given relation.

Note: it is not necessary for the rows corresponding to a given functional dependency to appear in *r*. A dependency can hold even if there are no rows in the relation, corresponding to it.

2.5. Normal forms

Exercise 47

The highest normal form is 1NF.

Exercise 48

The highest normal form is 3NF.

Exercise 49

R is not BCNF, thus there is a non-trivial dependency $X \to A$, where X is not a superkey. Since *X* is not a superkey, there is at least one attribute *B*, which is not defined by *X*. This means that $X \subseteq R \setminus AB$, as neither *A*, not *B* can be an element of X.
In dependency $X \to A$, let us add all those elements of $R \setminus AB$ to *X*, which are not already included in it. The so-obtained dependency is the exact same one that we have been looking for as we have been forming its left hand side until it became equal to $R \setminus AB$.
The obtained dependency is true, as *X* defines *A*, and this latter cannot be made false by adding further elements to the left hand side of the dependency.
Note: the last statement is a direct consequence of the expandability and the decomposition rule. If $P \to Q$ is true, then $PS \to Q$ is true as well, as we expand both sides with *S*, and then decompose the right hand side of the resulting functional dependency.

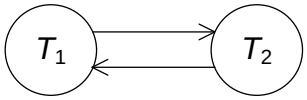
2.6. Transaction management

Exercise 62

The scheduling is serializable, its serial equivalent is: $T_2T_3T_1T_4$.

Exercise 63

The scheduling is not serializable, as there is no serial scheduling all effects of which are equal to those of our scheduling. The reason is as follows: At the end of the scheduling, it was T_1 who modified item *A* most recently, while it was T_2 who modified item *B* most recently. From among the possible serial schedules, in the case of T_1T_2, the last transaction to modify both data items is T_2. In the case of T_2T_1, the last transaction to modify both data items is T_1. The precedence graph of any legal schedule, in the case of any locking method (e.g. if we exchange each WRITE *X* command to a LOCK *X*, WRITE *X*, UNLOCK *X* series) will be as follows:



In case of 2PL: We know that if each transaction of a legal schedule follows 2PL, then the schedule is serializable. Thus, if a schedule is not serializable, there cannot be a legal schedule composed of 2PL transactions.

Exercise 65

The transaction is 2PL, but not strict. By exchanging the two rows, the transaction will be strict 2PL. The protocol ensures serializability and avoids cascading aborts.

LOCK *A*	synchronization
READ *A*	point
$A = A \times 2$	
COMMIT	commit point
WRITE *A*	writing over
UNLOCK *A*	

67. feladat

The schedule executes as shown below:

	T_1 $t(T_1) = 10$	T_2 $t(T_2) = 20$	R(*A*)	W(*A*)
(1)	READ *A*		10	0
(2)		WRITE *A*	10	20
(3)	WRITE *A*			

At a first glance, the schedule is not serializable as in step (3) WRITE *A* causes an abort, due to $t(T_1) < W(A)$.

Although, if transaction *T* wants to write item *A*, then in case of $R(A) \leq t(T) < W(A)$ the transaction does not necessarily have to be aborted. In this case, timestamps shall not be modified, and the item shall not be written. This is called *Thomas' Write Rule*.

This possibly surprising approach exploits the following observation: At the time of the attempted write operation, item *A* has already been written by a transaction *U*, which was started later ($t(T) < t(U)$). If in the future, a transaction *V* attempting to read *A* has a timestamp less than $W(A) = t(U)$, then *V* has to be aborted due to $t(V) < W(A)$. Also, if *V* has a timestamp greater than W(*A*), then *V* will have to read the value written by *U*. Thus, in neither of these cases will the value written by *T* be necessary.

It is very important that Thomas' Write Rule can only be used if the transaction with the greater timestamp has already committed. Think about what happens if in the above example, T_2 has further operations, and we try to apply the rule for T_1's write attempt on *A*, but before T_2 has committed. In this case, it is possible that after T_2 has committed, T_2 is aborted because of one of its later actions. In this case, *A* would have to hold the value that T_1 attempted to write into it. Thus value however, has not actually been written into item *A*, as the write operation was omitted.

One solution for this problem is assigning a C(*X*) commit bit to each item (*X*). C(*X*) is *true* by default, and is set to *false* when a write operation has been performed in the workspace, but the transaction has not yet committed. If C(*X*) = *false*, the further write and read operations on A have to wait until C(*X*) becomes *true* or until the last transaction writing *X* does not abort.

If the transaction that wrote X most recently – i.e. the transaction because of which C(X) is false – commits, the scheduler sets the C(X) bit to true. If this transaction aborts, both X and W(X) have to be reset to their previous values, and all transactions waiting for X have to repeat their read/write attempts. Using the commit bit eliminates the problems of dirty reads and the problem of inconsistency caused by Thomas' Write Rule, as this way we only apply the rule for "committed items", otherwise (in the case of C(X) = false) the transaction willing to write has to wait.

Note: The commit bit works the same way as locks used for timestamp scheduling.

If we use Thomas' Write Rule in the above example – that is, we do not modify timestamps, we omit the write operation in step (3), and do not abort the transaction – the *effect* of the resulting schedule will identical with that of serial schedule T_1, T_2, on any consistent database:

	T_1	T_2
(1)	READ A	
(2)		WRITE A
(3)	–	

≡

	T_1	T_2
(1)	READ A	
(2)	WRITE A	
(3)		WRITE A

Using Thomas' Write Rule we thus, found a "trick" by which we can eventually find serial equivalents to certain originally non-serializable schedules.