12. Relational Database Design

12.1. Introduction

The relational datamodel was introduced by Codd in 1970. It is the most widely used datamodel – extended with the possibilities of the World Wide Web –, because of its simplicity and flexibility. The main idea of the relational model is that data is organised in relational tables, where rows correspond to individual ***records*** and columns to ***attributes***. A ***relational schema*** consists of one or more relations and their attribute sets. In the present chapter only schemata consisting of one relation are considered for the sake of simplicity. In contrast to the mathematical concept of relations, in the relational schema the order of the attributes is not important, always ***sets*** of attributes are considered instead of ***lists***. Every attribute has an associated ***domain*** that is a set of elementary values that the attribute can take values from. As an example, consider the following schema.

Employee(Name,Mother's name,Social Security Number,Post,Salary)

The domain of attributes ***Name*** and ***Mother's name*** is the set of finite character strings (more precisely its subset containing all possible names). The domain of ***Social Security Number*** is the set of integers satisfying certain formal and parity check requirements. The attribute ***Post*** can take values from the set {Director,Section chief,System integrator,Programmer,Receptionist,Janitor,Handyman}. An ***instance*** of a schema R is a relation r if its columns correspond to the attributes of R and its rows contain values from the domains of attributes at the attributes' positions. A typical row of a relation of the Employee schema could be

(John Brown,Camille Parker,184-83-2010,Programmer,$172,000)

There can be dependencies between different data of a relation. For example, in an instance of the Employee schema the value of Social Security Number determines all other values of a row. Similarly, the pair (Name,Mother's name) is a unique identifier. Naturally, it may occur that some set of attributes do not determine all attributes of a record uniquely, just some of its subsets.

A relational schema has several ***integrity constraints*** attached. The most important kind of these is ***functional dependency***. Let U and V be two sets of attributes. V ***functionally depends*** on U, $U \to V$ in notation, means that whenever two records are identical in the

attributes belonging to U, then they must agree in the attribute belonging to V, as well. Throughout this chapter the attribute set $\{A_1, A_2, \ldots, A_k\}$ is denoted by $A_1A_2 \ldots A_k$ for the sake of convenience.

Example 12.1 *Functional dependencies* Consider the schema

R(**P**rofessor,**Su**bject,**R**oom,**St**udent,**G**rade,**T**ime).

The meaning of an individual record is that a given student got a given grade of a given subject that was taught by a given professor at a given time slot. The following functional dependencies are satisfied.

Su→P: One subject is taught by one professor.
PT→R: A professor teaches in one room at a time.
StT→R: A student attends a lecture in one room at a time.
StT→Su: A student attends a lecture of one subject at a time.
SuSt→G: A student receives a unique final grade of a subject.

In Example 12.1. the attribute set **StT** uniquely determines the values of all other attributes, furthermore it is minimal such set with respect to containment. This kind attribute sets are called ***keys***. If all attributes are functionally dependent on a set of attributes X, then X is called a ***superkey***. It is clear that every superkey contains a key and that any set of attributes containing a superkey is also a superkey.

12.2. Functional dependencies

Some functional dependencies valid for a given relational schema are known already in the design phase, others are consequences of these. The **StT→P** dependency is implied by the **StT→Su** and **Su→P** dependencies in Example 12.1.. Indeed, if two records agree on attributes **St** and **T**, then they must have the same value in attribute **Su**. Agreeing in **Su** and **Su→P** implies that the two records agree in **P**, as well, thus **StT→P** holds.

Definition 12.1 *Let R be a relational schema, F be a set of functional dependencies over R. The functional dependency $U \to V$ is* ***logically implied*** *by F, in notation $F \models U \to V$, if each instance of R that satisfies all dependencies of F also satisfies $U \to V$. The* ***closure*** *of a set F of functional dependencies is the set F^+ given by*

$$F^+ = \{U \to V \colon F \models U \to V\}.$$

12.2.1. Armstrong-axioms

In order to determine keys, or to understand logical implication between functional dependencies, it is necessary to know the closure F^+ of a set F of functional dependencies, or for a given $X \to Z$ dependency the question whether it belongs to F^+ must be decidable. For this, ***inference rules*** are needed that tell that from a set of functional dependencies what others follow. The ***Armstrong-axioms*** form a system of ***sound*** and ***complete*** inference rules. A system of rules is sound if only valid functional dependencies can be derived using it. It is complete, if every dependency $X \to Z$ that is logically implied by the set F is derivable

from F using the inference rules.

ARMSTRONG-AXIOMS

(A1) ***Reflexivity*** $Y \subseteq X \subseteq R$ implies $X \to Y$.

(A2) ***Augmentation*** If $X \to Y$, then for arbitrary $Z \subseteq R$, $XZ \to YZ$ holds.

(A3) ***Transitivity*** If $X \to Y$ and $Y \to Z$ hold, then $X \to Z$ holds, as well.

Example 12.2 *Derivation by the Armstrong-axioms* Let $R = ABCD$ and $F = \{A \to C, B \to D\}$, then AB is a key:

1. $A \to C$ is given.
2. $AB \to ABC$ 1. is augmented by (A2) with AB.
3. $B \to D$ is given.
4. $ABC \to ABCD$ 3. is augmented by (A2) with ABC.
5. $AB \to ABCD$ transitivity (A3) is applied to 2. and 4..

Thus it is shown that AB is superkey. That it is really a key, follows from algorithm CLOSURE(R, F, X).

There are other valid inference rules besides (A1)–(A3). The next lemma lists some, the proof is left to the Reader (Exercise 12.2-5.).

Lemma 12.2

1. ***Union rule*** $\{X \to Y, X \to Z\} \models X \to YZ$.
2. ***Pseudo transitivity*** $\{X \to Y, WY \to Z\} \models XW \to YZ$.
3. ***Decomposition*** *If* $X \to Y$ *holds and* $Z \subseteq Y$, *then* $X \to Z$ *holds, as well.*

The soundness of system (A1)–(A3) can be proven by easy induction on the length of the derivation. The completeness will follow from the proof of correctness of algorithm CLOSURE(R, F, X) by the following lemma. Let X^+ denote the ***closure*** of the set of attributes $X \subseteq R$ with respect to the family of functional dependencies F, that is $X^+ = \{A \in R\colon X \to A$ follows from F by the Armstrong-axioms$\}$.

Lemma 12.3 *The functional dependency* $X \to Y$ *follows from the family of functional dependencies* F *by the Armstrong-axioms iff* $Y \subseteq X^+$.

Proof. Let $Y = A_1 A_2 \ldots A_n$ where A_i's are attributes, and assume that $Y \subseteq X^+$. $X \to A_i$ follows by the Armstrong-axioms for all i by the definition of X^+. Applying the union rule of Lemma 12.2 $X \to Y$ follows. On the other hand, assume that $X \to Y$ can be derived by the Armstrong-axioms. By the decomposition rule of Lemma 12.2 $X \to A_i$ follows by (A1)–(A3) for all i. Thus, $Y \subseteq X^+$. ■

12.2.2. Closures

Calculation of closures is important in testing equivalence or logical implication between systems of functional dependencies. The first idea could be that for a given family F of functional dependencies in order to decide whether $F \models \{X \to Y\}$, it is enough to calculate F^+ and check whether $\{X \to Y\} \in F^+$ holds. However, the size of F^+ could be exponential

in the size of input. Consider the family F of functional dependencies given by

$$F = \{A \to B_1, A \to B_2, \ldots, A \to B_n\}.$$

F^+ consists of all functional dependencies of the form $A \to Y$, where $Y \subseteq \{B_1, B_2, \ldots, B_n\}$, thus $|F^+| = 2^n$. Nevertheless, the closure X^+ of an attribute set X with respect to F can be determined in linear time of the total length of functional dependencies in F. The following is an algorithm that calculates the closure X^+ of an attribute set X with respect to F. The input consists of the schema R, that is a finite set of attributes, a set F of functional dependencies defined over R, and an attribute set $X \subseteq R$.

CLOSURE(R,F,X)

```
 1  X^(0) ← X
 2  i ← 0
 3  G ← F                                  ▷ Functional dependencies not used yet.
 4  repeat
 5      X^(i+1) ← X^(i)
 6      for all Y → Z in G
 7          do if Y ⊆ X^(i)
 8              then X^(i+1) ← X^(i+1) ∪ Z
 9                   G ← G \ {Y → Z}
10      i ← i + 1
11  until X^(i−1) = X^(i)
```

It is easy to see that the attributes that are put into any of the $X^{(j)}$'s by CLOSURE(R,F,X) really belong to X^+. The harder part of the correctness proof of this algorithm is to show that each attribute belonging to X^+ will be put into some of the $X^{(j)}$'s.

Theorem 12.4 CLOSURE(R,F,X) *correctly calculates* X^+.

Proof. First we prove by induction that if an attribute A is put into an $X^{(j)}$ during CLOSURE(R,F,X), then A really belongs to X^+.
Base case: $j = 0$. I this case $A \in X$ and by reflexivity (A1) $A \in X^+$.
Induction step: Let $j > 0$ and assume that $X^{(j-1)} \subseteq X^+$. A is put into $X^{(j)}$, because there is a functional dependency $Y \to Z$ in F, where $Y \subseteq X^{(j-1)}$ and $A \in Z$. By induction, $Y \subseteq X^+$ holds, which implies using Lemma 12.3 that $X \to Y$ holds, as well. By transitivity (A3) $X \to Y$ and $Y \to Z$ implies $X \to Z$. By reflexivity (A1) and $A \in Z$, $Z \to A$ holds. Applying transitivity again, $X \to A$ is obtained, that is $A \in X^+$.

On the other hand, we show that if $A \in X^+$, then A is contained in the result of CLOSURE(R,F,X). Suppose in contrary that $A \in X^+$, but $A \notin X^{(i)}$, where $X^{(i)}$ is the result of CLOSURE(R,F,X). By the stop condition in line 9 this means $X^{(i)} = X^{(i+1)}$. An instance r of the schema R is constructed that satisfies every functional dependency of F, but $X \to A$ does not hold in r if $A \notin X^{(i)}$. Let r be the following two-rowed relation:

Attributes of $X^{(i)}$				Other attributes			
1	1	…	1	1	1	…	1
1	1	…	1	0	0	…	0

Let us suppose that the above r violates a $U \to V$ functional dependency of F, that is $U \subseteq X^{(i)}$, but V is not a subset of $X^{(i)}$. However, in this case Closure(R,F,X) could not have stopped yet, since $X^{(i)} \neq X^{(i+1)}$.

$A \in X^+$ implies using Lemma 12.3 that $X \to A$ follows from F by the Armstrong-axioms. (A1)–(A3) is a sound system of inference rules, hence in every instance that satisfies F, $X \to A$ must hold. However, the only way this could happen in instance r is if $A \in X^{(i)}$. ■

Let us observe that the relation instance r given in the proof above provides the completeness proof for the Armstrong-axioms, as well. Indeed, the closure X^+ calculated by Closure(R,F,X) is the set of those attributes for which $X \to A$ follows from F by the Armstrong-axioms. Meanwhile, for every other attribute B, there exist two rows of r that agree on X, but differ in B, that is $F \models X \to B$ ***does not*** hold.

The running tome of Closure(R,F,X) is $O(n^2)$, where n is the length of he input. Indeed, in the **repeat – until** loop of lines 4–11 every not yet used dependency is checked, and the body of the loop is executed at most $|R \setminus X| + 1$ times, since it is started again only if $X^{(i-1)} \neq X^{(i)}$, that is a new attribute is added to the closure of X. However, the running time can be reduced to linear with appropriate bookkeeping.

1. For every yet unused $W \to Z$ dependency of F it is kept track of how many attributes of W are not yet included in the closure ($i[W,Z]$).

2. For every attribute A those yet unused dependencies are kept in a doubly linked list L_A whose left side contains A.

3. Those not yet used dependencies $W \to Z$ are kept in a linked list J, whose left side W's every attribute is contained in the closure already, that is for which $i[W,Z] = 0$.

It is assumed that the family of functional dependencies F is given as a set of attribute pairs (W,Z), representing $W \to Z$. The Linear-Closure(R,F,X) algorithm is a modification of Closure(R,F,X) using the above bookkeeping, whose running time is linear. R is the schema, F is the given family of functional dependencies, and we are to determine the closure of attribute set X.

Algorithm Linear-Closure(R,F,X) consists of two parts. In the initialisation phase (lines 1–13) the lists are initialised. The loops of lines 2–5 and 6–8, respectively, take $O(\sum_{(W,Z)\in F} |W|)$ time. The loop in lines 9–11 means $O(|F|)$ steps. If the length of the input is denoted by n, then this is $O(n)$ steps altogether.

During the execution of lines 14–23, every functional dependency (W,Z) is examined at most once, when it is taken off from list J. Thus, lines 15–16 and 23 take at most $|F|$ steps. The running time of the loops in line 17–22 can be estimated by observing that the sum $\sum i[W,Z]$ is decreased by one in each execution, hence it takes $O(\sum i_0[W,Z])$ steps, where $i_0[W,Z]$ is the $i[W,Z]$ value obtained in the initialisation phase. However, $\sum i_0[W,Z] \leq \sum_{(W,Z)\in F} |W|$, thus lines 14–23 also take $O(n)$ time in total.

```
LINEAR-CLOSURE(R,F,X)
 1  ▷ Initialisation phase.
 2  for all (W,Z) ∈ F
 3      do for all A ∈ W
 4             do add (W,Z) to list L_A
 5      i[W,Z] ← 0
 6  for all A ∈ R \ X
 7      do for all (W,Z) of list L_A
 8             do i[W,Z] ← i[W,Z] + 1
 9  for all (W,Z) ∈ F
10      do if i[W,Z] = 0
11            then add (W,Z) to list J
12  X⁺ ← X
13  ▷ End of initialisation phase.
14  while J is nonempty
15        do (W,Z) ← head(J)
16           delete (W,Z) from list J
17           for all A ∈ Z \ X⁺
18               do for all (W,Z) of list L_A
19                      do i[W,Z] ← i[W,Z] − 1
20                         if i[W,Z] = 0
21                            then add (W,Z) to list J
22                                 delete (W,Z)from list L_A
23        X⁺ ← X⁺ ∪ Z
24  return X⁺
```

12.2.3. Minimal cover

Algorithm LINEAR-CLOSURE(R,F,X) can be used to test equivalence of systems of dependencies. Let F and G be two families of functional dependencies. F and G are said to be ***equivalent,*** if exactly the same functional dependencies follow from both, that is $F^+ = G^+$. It is clear that it is enough to check for all functional dependencies $X \to Y$ in F whether it belongs to G^+, and vice versa, for all $W \to Z$ in G, whether it is in F^+. Indeed, if some of these is not satisfied, say $X \to Y$ is not in G^+, then surely $F^+ \neq G^+$. On the other hand, if all $X \to Y$ are in G^+, then a proof of a functional dependency $U \to V$ from F^+ can be obtained from dependencies in G in such a way that to the derivation of the dependencies $X \to Y$ of F from G, the derivation of $U \to V$ from F is concatenated. In order to decide that a dependency $X \to Y$ from F is in G^+, it is enough to construct the closure $X^+(G)$ of attribute set X with respect to G using LINEAR-CLOSURE(R,G,X), then check whether $Y \subseteq X^+(G)$ holds. The following special functional dependency system equivalent with F is useful.

Definition 12.5 *The system of functional dependencies G is a **minimal cover** of the family of functional dependencies F iff G is equivalent with F, and*

1. *functional dependencies of G are in the form $X \to A$, where A is an attribute and $A \notin X$,*
2. *no functional dependency can be dropped from G, i.e., $(G - \{X \to A\})^+ \subsetneq G^+$,*
3. *the left sides of dependencies in G are minimal, that is $X \to A \in G$, $Y \subsetneq X \Longrightarrow ((G - \{X \to A\}) \cup \{Y \to A\})^+ \neq G^+$.*

Every set of functional dependencies have a minimal cover, namely algorithm MINIMAL-COVER(R, F) constructs one.

MINIMAL-COVER(R, F)

```
1  G ← ∅
2  for all X → Y ∈ F
3      do for all A ∈ Y − X
4             do G ← G ∪ X → A
5  ▷ Each right hand side consists of a single attribute.
6  for all X → A ∈ G
7      do while there exists B ∈ X: A ∈ (X − B)⁺(G)
8             X ← X − B
9  ▷ Each left hand side is minimal.
10 for all X → A ∈ G
11     do if A ∈ X⁺(G − {X → A})
12           then G ← G − {X → A}
13 ▷ No redundant dependency exists.
```

After executing the loop of lines 2–4, the right hand side of each dependency in G consists of a single attribute. The equality $G^+ = F^+$ follows from the union rule of Lemma 12.2 and the reflexivity axiom. Lines 6–8 minimise the left hand sides. In line 11 it is checked whether a given functional dependency of G can be removed without changing the closure. $X^+(G - \{X \to A\})$ is the closure of attribute set X with respect to the family of functional dependencies $G - \{X \to A\}$.

Proposition 12.6 MINIMAL-COVER(R, F) *calculates a minimal cover of F.*

Proof. It is enough to show that during execution of the loop in lines 10–12, no functional dependency $X \to A$ is generated whose left hand side could be decreased. Indeed, if a $X \to A$ dependency would exist, such that for some $Y \subsetneq X$ $Y \to A \in G^+$ held, then $Y \to A \in G'^+$ would also hold, where G' is the set of dependencies considered when $X \to A$ is checked in lines 6–8. $G \subseteq G'$, which implies $G^+ \subseteq G'^+$ (see Exercise 12.2-1.). Thus, X should have been decreased already during execution of the loop in lines 6–8. ■

12.2.4. Keys

In database design it is important to identify those attribute sets that uniquely determine the data in individual records.

Definition 12.7 *Let (R, F) be a relational schema. The set of attributes $X \subseteq R$ is called a* ***superkey,*** *if $X \to R \in F^+$. A superkey X is called a* ***key,*** *if it is minimal with respect to containment, that is no proper subset $Y \subsetneq X$ is key.*

The question is how the keys can be determined from (R, F)? What makes this problem hard is that the number of keys could be super exponential function of the size of (R, F). In particular, Yu and Johnson constructed such relational schema, where $|F| = k$, but the number of keys is $k!$. Békéssy and Demetrovics gave a beautiful and simple proof of the fact that starting from k functional dependencies, at most $k!$ key can be obtained. (This was independently proved by Osborne and Tompa.)

The proof of Békéssy and Demetrovics is based on the operation $*$ they introduced, which is defined for functional dependencies.

Definition 12.8 *Let $e_1 = U \to V$ and $e_2 = X \to Y$ be two functional dependencies. The binary operation $*$ is defined by*

$$e_1 * e_2 = U \cup ((R - V) \cap X) \to V \cup Y.$$

Some properties of operation $*$ is listed, the proof is left to the Reader (Exercise 12.2-3.). Operation $*$ is associative, furthermore it is idempotent in the sense that if $e = e_1 * e_2 * \cdots * e_k$ and $e' = e * e_i$ for some $1 \leq i \leq k$, then $e' = e$.

Proposition 12.9 (Békéssy and Demetrovics). *Let (R, F) be a relational schema and let $F = \{e_1, e_2, \ldots, e_k\}$ be a listing of the functional dependencies. If X is a key, then $X \to R = e_{\pi_1} * e_{\pi_2} * \ldots * e_{\pi_s} * d$, where $(\pi_1, \pi_2, \ldots, \pi_s)$ is an ordered subset of the index set $\{1, 2, \ldots, k\}$, and d is a trivial dependency in the form $D \to D$.*

Proposition 12.9 bounds in some sense the possible sets of attributes in the search for keys. The next proposition gives lower and upper bounds for the keys.

Proposition 12.10 *Let (R, F) be a relational schema and let $F = \{U_i \to V_i : 1 \leq i \leq k\}$. Let us assume without loss of generality that $U_i \cap V_i = \emptyset$. Let $\mathcal{U} = \bigcup_{i=1}^{k} U_i$ and $\mathcal{V} = \bigcup_{i=1}^{k} V_i$. If K is a key in the schema (R, F), then*

$$\mathcal{H}_L = R - \mathcal{V} \subseteq K \subseteq (R - \mathcal{V}) \cup \mathcal{U} = \mathcal{H}_U.$$

The proof is not too hard, it is left as an exercise for the Reader (Exercise 12.2-4.). The algorithm List-keys(R, F) that lists the keys of the schema (R, F) is based on the bounds of Proposition 12.10. The running time can be bounded by $O(n!)$, but one cannot expect any better, since to list the output needs that much time in worst case.

LIST-KEYS(R, F)

```
1  ▷ Let 𝒰 and 𝒱 be as defined in Proposition 12.10
2  if 𝒰 ∩ 𝒱 = ∅
3     then return R − 𝒱
4  ▷ R − 𝒱 is the only key.
5  if (R − 𝒱)⁺ = R
6     then return R − 𝒱
7     ▷ R − 𝒱 is the only key.
8  𝒦 ← ∅
9  for all permutations A_1, A_2, ... A_h of the attributes of 𝒰 ∩ 𝒱
10    do K ← (R − 𝒱) ∪ 𝒰
11       for i ← 1 to h
12          do Z ← K − A_i
13             if Z⁺ = R
14                then K ← Z
15       𝒦 ← 𝒦 ∪ {K}
16 return 𝒦
```

Exercises

12.2-1 Let R be a relational schema and let F and G be families of functional dependencies over R.Show that

a. $F \subseteq F^+$.

b. $(F^+)^+ = F^+$.

c. If $F \subseteq G$, then $F^+ \subseteq G^+$.

Formulate and prove similar properties of the closure X^+ – with respect to F – of an attribute set X.

12.2-2 Derive the functional dependency $AB \to F$ from the set of dependencies $G = \{AB \to C, A \to D, CD \to EF\}$ using Armstrong-axioms (A1)–(A3).

12.2-3 Show that operation $*$ is associative, furthermore if for functional dependencies $e_1, e_2, \ldots, e_k$ we have $e = e_1 * e_2 * \cdots * e_k$ and $e' = e * e_i$ for some $1 \leq i \leq k$, then $e' = e$.

12.2-4 Prove Proposition 12.10.

12.2-5 Prove the union, pseudo transitivity and decomposition rules of Lemma 12.2.

12.3. Decomposition of relational schemata

A ***decomposition*** of a relational schema $R = \{A_1, A_2, \ldots, A_n\}$ is a collection $\rho = \{R_1, R_2, \ldots, R_k\}$ of subsets of R such that

$$R = R_1 \cup R_2 \cup \cdots \cup R_k.$$

The R_i's need not be disjoint, in fact in most application they must not be. One important motivation of decompositions is to avoid ***anomalies***.

Example 12.3 *Anomalies* Consider the following schema

SUPPLIER-INFO(SNAME,ADDRESS,ITEM,PRICE)

This schema encompasses the following problems:

1. ***Redundancy.*** The address of a supplier is recorded with every item it supplies.
2. ***Possible inconsistency (update anomaly).*** As a consequence of redundancy, the address of a supplier might be updated in some records and might not be in some others, hence the supplier would not have a unique address, even though it is expected to have.
3. ***Insertion anomaly.*** The address of a supplier cannot be recorded if it does not supply anything at the moment. One could try to use `NULL` values in attributes ITEM and PRICE, but would it be remembered that it must be deleted, when a supplied item is entered for that supplier? More serious problem that SNAME and ITEM together form a key of the schema, and the `NULL` values could make it impossible to search by an index based on that key.
4. ***Deletion anomaly*** This is the opposite of the above. If all items supplied by a supplier are deleted, then as a side effect the address of the supplier is also lost.

All problems mentioned above are eliminated if schema SUPPLIER-INFO is replaced by two subschemata:

SUPPLIER(SNAME,ADDRESS),
SUPPLIES(SNAME,ITEM,PRICE).

In this case each suppliers address is recorded only once, and it is not necessary that the supplier supplies a item in order its address to be recorded. For the sake of convenience the attributes are denoted by single characters S (SNAME), A (ADDRESS), I (ITEM), P (PRICE).

Question is that is it correct to replace the schema $SAIP$ by SA and SIP? Let r be and instance of schema $SAIP$. It is natural to require that if SA and SIP is used, then the relations belonging to them are obtained projecting r to SA and SIP, respectively, that is $r_{SA} = \pi_{SA}(r)$ and $r_{SIP} = \pi_{SIP}(r)$. r_{SA} and r_{SIP} contains the same information as r, if r can be reconstructed using only r_{SA} and r_{SIP}. The calculation of r from r_{SA} and r_{SIP} can bone by the ***natural join*** operator.

Definition 12.11 *The **natural join** of relations r_i of schemata R_i $(i = 1, 2, \ldots n)$ is the relation s belonging to the schema $\cup_{i=1}^{n} R_i$, which consists of all rows μ that for all i there exists a row ν_i of relation r_i such that $\mu[R_i] = \nu_i[R_i]$. In notation $s = \bowtie_{i=1}^{n} r_i$.*

Example 12.4 Let $R_1 = AB$, $R_2 = BC$, $r_1 = \{ab, a'b', ab''\}$ and $r_2\{bc, bc', b'c''\}$. The natural join of r_1 and r_2 belongs to the schema $R = ABC$, and it is the relation $r_1 \bowtie r_2 = \{abc, abc', a'b'c''\}$.

If s is the natural join of r_{SA} and r_{SIP}, that is $s = r_{SA} \bowtie r_{SIP}$, then $\pi_{SA}(s) = r_{SA}$ és $\pi_{SIP}(s) = r_{SIP}$ by Lemma 12.12. If $r \neq s$, then the original relation could not be reconstructed knowing only r_{SA} and r_{SIP}.

12.3.1. Lossless join

Let $\rho = \{R_1, R_2, \ldots, R_k\}$ be a decomposition of schema R, furthermore let F be a family of functional dependencies over R. The decomposition ρ is said to have ***lossless join property*** (with respect to F), if every instance r of R that satisfies F also satisfies

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \cdots \bowtie \pi_{R_k}(r).$$

That is, relation r is the natural join of its projections to attribute sets R_i, $i = 1, 2, \ldots, k$. For a decomposition $\rho = \{R_1, R_2, \ldots, R_k\}$, let m_ρ denote the the mapping which assigns to relation r the relation $m_\rho(r) = \bowtie_{i=1}^{k} \pi_{R_i}(r)$. Thus, the lossless join property with respect to a family of functional dependencies means that $r = m_\rho(r)$ for all instances r that satisfy F.

Lemma 12.12 *Let $\rho = \{R_1, R_2, \ldots, R_k\}$ be a decomposition of schema R, and let r be an arbitrary instance of R. Furthermore, let $r_i = \pi_{R_i}(r)$. Then*

1. $r \subseteq m_\rho(r)$.

2. *If $s = m_\rho(r)$, then $\pi_{R_i}(s) = r_i$.*

3. $m_\rho(m_\rho(r)) = m_\rho(r)$.

The proof is left to the Reader (Exercise 12.3-7.).

12.3.2. Checking the lossless join property

It is relatively not hard to check that a decomposition $\rho = \{R_1, R_2, \ldots, R_k\}$ of schema R has the lossless join property. The essence of algorithm JOIN-TEST(R,F,ρ) is the following.
A $k \times n$ array T is constructed, whose column j corresponds to attribute A_j, while row i corresponds to schema R_i. $T[i, j] = 0$ if $A_j \in R_i$, otherwise $T[i, j] = i$.

The following step is repeated until there is no more possible change in the array. Consider a functional dependency $X \to Y$ from F. If a pair of rows i and j agree in all attributes of X, then their values in attributes of Y are made equal. More precisely, if one of the values in an attribute of Y is 0, then the other one is set to 0, as well, otherwise it is arbitrary which of the two values is set to be equal to the other one. If a symbol is changed, then ***each*** of its occurrences in that column must be changed accordingly. If at the end of this process there is an all 0 row in T, then the decomposition has the lossless join property, otherwise, it is lossy.

JOIN-TEST(R,F,ρ)

```
 1 ▷ Initialisation phase.
 2 for i ← 1 to |ρ|
 3     do for j ← 1 to |R|
 4           do if A_j ∈ R_i
 5                 then T[i, j] ← 0
 6                 else T[i, j] ← i
 7 ▷ End of initialisation phase.
 8 S ← T
 9 repeat
10       T ← S
11       for all {X → Y} ∈ F
12            do for i ← 1 to |ρ| − 1
13                  do for j ← i + 1 to |R|
14                        do if for all A_h in X (S[i, h] = S[j, h])
15                              then EQUATE(i, j, S, Y)
16 until S = T
17 if there exist an all 0 row in S
18    then return "Lossless join"
19    else return "Lossy join"
```

Procedure EQUATE(i, j, S, Y) makes the appropriate symbols equal.

EQUATE(i, j, S, Y)

```
 1 for A_l ∈ Y
 2     do if S[i, l] · S[j, l] = 0
 3           then
 4                for d ← 1 to k
 5                     do if S[d, l] = S[i, l] ∨ S[d, l] = S[j, l]
 6                           then S[d, l] ← 0
 7           else
 8                for d ← 1 to k
 9                     do if S[d, l] = S[j, l]
10                           then S[d, l] ← S[i, l]
```

Example 12.5 *Checking lossless join property* Let $R = ABCDE$, $R_1 = AD$, $R_2 = AB$, $R_3 = BE$, $R_4 = CDE$, $R_5 = AE$, furthermore let the functional dependencies be $\{A \to C, B \to C, C \to D, DE \to C, CE \to A\}$. The initial array is shown on Figure 12.1(a). Using $A \to C$ values 1,2,5 in column C can be equated to 1. Then applying $B \to C$ value 3 of column C again can be changed to 1. The result is shown on Figure 12.1(b). Now $C \to D$ can be used to change values 2,3,5 of column D to 0. Then applying $DE \to C$ (the only nonzero) value 1 of column C can be set to 0. Finally, $CE \to A$ makes it possible to change values 3 and 4 in column A to be changed to 0. The final result is shown on Figure 12.1(c). The third row consists of only zeroes, thus the decomposition has the lossless join property.

It is clear that the running time of algorithm JOIN-TEST(R,F,ρ) is polynomial in the length

A	B	C	D	E
0	1	1	0	1
0	0	2	2	2
3	0	3	3	0
4	4	0	0	0
0	5	5	5	0

(a)

A	B	C	D	E
0	1	1	0	1
0	0	1	2	2
3	0	1	3	0
4	4	0	0	0
0	5	1	5	0

(b)

A	B	C	D	E
0	1	0	0	1
0	0	0	2	2
0	0	0	0	0
0	4	0	0	0
0	5	0	0	0

(c)

Figure 12.1. Application of JOIN-TEST(R,F,ρ).

of the input. The important thing is that it uses only the schema, not the instance r belonging to the schema. Since the size of an instance is larger than the size of the schema by many orders of magnitude, the running time of an algorithm using the schema only is negligible with respect to the time required by an algorithm processing the data stored.

Theorem 12.13 *Procedure* JOIN-TEST(R,F,ρ) *correctly determines whether a given decomposition has the lossless join property.*

Proof. Let us assume first that the resulting array T contains no all zero row. T itself can be considered as a relational instance over the schema R. This relation satisfies all functional dependencies from F, because the algorithm finished since there was no more change in the table during checking the functional dependencies. It is true for the starting table that its projections to every R_i's contain an all zero row, and this property does not change during the running of the algorithm, since a 0 is never changed to another symbol. It follows, that the natural join $m_\rho(T)$ contains the all zero row, that is $T \neq m_\rho(T)$. Thus the decomposition is lossy. The proof of the other direction is only sketched.

Logic, domain calculus is used. The necessary definitions can be found in the books of Abiteboul, Hull and Vianu, or Ullman, respectively. Imagine that variable a_j is written in place of zeroes, and b_{ij} is written in place of i's in column j, and JOIN-TEST(R,F,ρ) is run in this setting. The resulting table contains row $a_1 a_2 \dots a_n$, which corresponds to the all zero row. Every table can be viewed as a shorthand notation for the following domain calculus expression

$$\{a_1 a_2 \dots a_n \mid (\exists b_{11}) \dots (\exists b_{kn})\,(R(w_1) \wedge \dots \wedge R(w_k))\}, \tag{12.1}$$

where w_i is the ith row of T. If T is the starting table, then formula (12.1) defines m_ρ exactly. As a justification note that for a relation r, $m_\rho(r)$ contains the row $a_1 a_2 \dots a_n$ iff r contains for all i a row whose jth coordinate is a_j if A_j is an attribute of R_i, and arbitrary values represented by variables b_{il} in the other attributes.

Consider an arbitrary relation r belonging to schema R that satisfies the dependencies of F. The modifications (equating symbols) of the table done by JOIN-TEST(R,F,ρ) do not change the set of rows obtained from r by (12.1), if the modifications are done in the formula, as well. Intuitively it can be seen from the fact that only such symbols are equated in (12.1), that can only take equal values in a relation satisfying functional dependencies of F. The exact proof is omitted, since it is quiet tedious.

Since in the result table of JOIN-TEST(R,F,ρ) the all a's row occurs, the domain calculus formula that belongs to this table is of the following form:

$$\{a_1 a_2 \dots a_n \mid (\exists b_{11}) \dots (\exists b_{kn}) \, (R(a_1 a_2 \dots a_n) \wedge \cdots)\} . \tag{12.2}$$

It is obvious that if (12.2) is applied to relation r belonging to schema R, then the result will be a subset of r. However, if r satisfies the dependencies of F, then (12.2) calculates $m_\rho(r)$. According to Lemma 12.12, $r \subseteq m_\rho(r)$ holds, thus if r satisfies F, then (12.2) gives back r exactly, so $r = m_\rho(r)$, that is the decomposition has the lossless join property. ■

Procedure JOIN-TEST(R,F,ρ) can be used independently of the number of parts occurring in the decomposition. The price of this generality is paid in the running time requirement. However, if R is to be decomposed only into ***two*** parts, then CLOSURE(R,F,X) or LINEAR-CLOSURE(R,F,X) can be used to obtain the same result faster, according to the next theorem.

Theorem 12.14 *Let $\rho = (R_1, R_2)$ be a decomposition of R, furthermore let F be a set of functional dependencies. Decomposition ρ has the lossless join property with respect to F iff*

$$(R_1 \cap R_2) \to (R_1 - R_2) \text{ or } (R_1 \cap R_2) \to (R_2 - R_1) .$$

These dependencies need not be in F, it is enough if they are in F^+.

Proof. The starting table in procedure JOIN-TEST(R,F,ρ) is the following:

	$R_1 \cap R_2$	$R_1 - R_2$	$R_2 - R_1$
row of R_1	$00\dots0$	$00\dots0$	$11\dots1$
row of R_2	$00\dots0$	$22\dots2$	$00\dots0$

(12.3)

It is not hard to see using induction on the number of steps done by JOIN-TEST(R,F,ρ) that if the algorithm changes both values of the column of an attribute A to 0, then $A \in (R_1 \cap R_2)^+$. This is obviously true at the start. If at some time values of column A must be equated, then by lines 11–14 of the algorithm, there exists $\{X \to Y\} \in F$, such that the two rows of the table agree on X, and $A \in Y$. By the induction assumption $X \subseteq (R_1 \cap R_2)^+$ holds. Applying Armstrong-axioms (transitivity and reflexivity), $A \in (R_1 \cap R_2)^+$ follows.

On the other hand, let us assume that $A \in (R_1 \cap R_2)^+$, that is $(R_1 \cap R_2) \to A$. Then this functional dependency can be derived from F using Armstrong-axioms. By induction on the length of this derivation it can be seen that procedure JOIN-TEST(R,F,ρ) will equate the two values of column A, that is set them to 0. Thus, the row of R_1 will be all 0 iff $(R_1 \cap R_2) \to (R_2 - R_1)$, similarly, the row of R_2 will be all 0 iff $(R_1 \cap R_2) \to (R_1 - R_2)$. ■

12.3.3. Dependency preserving decompositions

The lossless join property is important so that a relation can be recovered from its projections. In practice, usually not the relation r belonging to the underlying schema R is stored, but relations $r_i = r[R_i]$ for an appropriate decomposition $\rho = (R_1, R_2, \ldots, R_k)$, in order to avoid anomalies. The functional dependencies F of schema R are ***integrity constraints*** of the database, relation r is consistent if it satisfies all prescribed functional dependencies. When during the life time of the database updates are executed, that is rows are inserted into or deleted from the projection relations, then it may happen that the natural join of the new projections does not satisfy the functional dependencies of F. It would be too costly to join the projected relations – and then project them again – after each update to check the integrity constraints. However, the ***projection*** of the family of functional dependencies F to an attribute set Z can be defined: $\pi_Z(F)$ consists of those functional dependencies $\{X \to Y\} \in F^+$, where $XY \subseteq Z$. After an update, if relation r_i is changed, then it is relatively easy to check whether $\pi_{R_i}(F)$ still holds. Thus, it would be desired if family F would be logical implication of the families of functional dependencies $\pi_{R_i}(F)$ $i = 1, 2, \ldots, k$. Let $\pi_\rho(F) = \bigcup_{i=1}^{k} \pi_{R_i}(F)$.

Definition 12.15 *The decomposition ρ is said to be **dependency preserving**, if*

$$\pi_\rho(F)^+ = F^+.$$

Note that $\pi_\rho(F) \subseteq F^+$, hence $\pi_\rho(F)^+ \subseteq F^+$ always holds. Consider the following example.

Example 12.6 Let R = (**C**ity,**S**treet,**Z**ip code) be the underlying schema, furthermore let $F = \{CS \to Z, Z \to C\}$ be the functional dependencies. Let the decomposition ρ be $\rho = (CZ, SZ)$. This has the lossless join property by Theorem 12.14. $\pi_\rho(F)$ consists of $Z \to C$ besides the trivial dependencies. Let $R_1 = CZ$ and $R_2 = SZ$. Two rows are inserted into each of the projections belonging to schemata R_1 and R_2, respectively, so that functional dependencies of the projections are satisfied:

R_1	C	Z
	Fort Wayne	46805
	Fort Wayne	46815

R_2	S	Z
	Coliseum Blvd	46805
	Coliseum Blvd	46815

In this case R_1 and R_2 satisfy the dependencies prescribed for them separately, however in $R_1 \bowtie R_2$ the dependency $CS \to Z$ does not hold.

It is true as well, that none of the decompositions of this schema preserves the dependency $CS \to Z$. Indeed, this is the only dependency that contains Z on the right hand side, thus if it is to be preserved, then there has to be a subschema that contains C, S, Z, but then the decomposition would not be proper. This will be considered again when decomposition into normal forms is treated.

Note that it may happen that decomposition ρ preserves functional dependencies, but does not have the lossless join property. Indeed, let $R = ABCD$, $F = \{A \to B, C \to D\}$, and let the decomposition be $\rho = (AB, CD)$.

Theoretically it is very simple to check whether a decomposition $\rho = (R_1, R_2, \ldots R_k)$ is dependency preserving. Just F^+ needs to be calculated, then projections need to be taken, finally one should check whether the union of the projections is equivalent with F. The main problem with this approach is that even calculating F^+ may need exponential time.

Nevertheless, the problem can be solved without explicitly determining F^+. Let $G = \pi_\rho(F)$. G will not be calculated, only its equivalence with F will be checked. For this end,

it needs to be decidable for all functional dependencies $\{X \to Y\} \in F$ that if X^+ is taken with respect to G, whether it contains Y. The trick is that X^+ is determined ***without*** full knowledge of G by repeatedly taking the effect to the closure of the projections of F onto the individual R_i's. That is, the concept of S-operation on an attribute set Z is introduced, where S is another set of attributes: Z is replaced by $Z \cup ((Z \cap S)^+ \cap S)$, where the closure is taken with respect to F. Thus, the closure of the part of Z that lies in S is taken with respect to F, then from the resulting attributes those are added to Z, which also belong to S.

It is clear that the running time of algorithm PRESERVE(ρ, F) is polynomial in the length of the input. More precisely, the outermost **for** loop is executed at most once for each dependency in F (it may happen that it turns out earlier that some dependency is not preserved). The body of the **repeat–until** loop in lines 3–7. requires linear number of steps, it is executed at most $|R|$ times. Thus, the body of the **for** loop needs quadratic time, so the total running time can be bounded by the cube of the input length.

PRESERVE(ρ, F)

```
 1 for all (X → Y) ∈ F
 2     do Z ← X
 3        repeat
 4              W ← Z
 5              for i ← 1 to k
 6                  do Z ← Z ∪ (LINEAR-CLOSURE(R, F, Z ∩ R_i) ∩ R_i)
 7        until Z = W
 8     if Y ⊈ Z
 9        then return "Not dependency preserving"
10 return "Dependency preserving"
```

Example 12.7 Consider the schema $R = ABCD$, let the decomposition be $\rho = \{AB, BC, CD\}$, and dependencies be $F = \{A \to B, B \to C, C \to D, D \to A\}$. That is, by the visible cycle of the dependencies, every attribute determines all others. Since D and A do not occur together in the decomposition one might think that the dependency $D \to A$ is not preserved, however this intuition is wrong. The reason is that during the projection to AB, not only the dependency $A \to B$ is obtained, but $B \to A$, as well, since not F, but F^+ is projected. Similarly, $C \to B$ and $D \to C$ are obtained, as well, but $D \to A$ is a logical implication of these by the transitivity of the Armstrong axioms. Thus it is expected that PRESERVE(ρ, F) claims that $D \to A$ is preserved.

Start from the attribute set $Y = \{D\}$. There are three possible operations, the AB-operation, the BC-operation and the CD-operation. The first two obviously does not add anything to $\{D\}^+$, since $\{D\} \cap \{A, B\} = \{D\} \cap \{B, C\} = \emptyset$, that is the closure of the empty set should be taken, which is empty (in the present example). However, using the CD-operation:

$$\begin{aligned} Z &= \{D\} \cup ((\{D\} \cap \{C, D\})^+ \cap \{C, D\}) \\ &= \{D\} \cup (\{D\}^+ \cap \{C, D\}) \\ &= \{D\} \cup (\{A, B, C, D\} \cap \{C, D\}) \\ &= \{C, D\}. \end{aligned}$$

In the next round using the BC-operation the actual $Z = \{C, D\}$ is changed to $Z = \{B, C, D\}$, finally applying the AB-operation on this, $Z = \{A, B, C, D\}$ is obtained. This cannot change, so procedure

PRESERVE(ρ, F) stops. Thus, with respect to the family of functional dependencies

$$G = \pi_{AB}(F) \cup \pi_{BC}(F) \cup \pi_{CD}(F),$$

$\{D\}^+ = \{A, B, C, D\}$ holds, that is $G \models D \to A$. It can be checked similarly that the other dependencies of F are in G^+ (as a fact in G).

Theorem 12.16 *The procedure* PRESERVE(ρ, F) *determines correctly whether the decomposition ρ is dependency preserving.*

Proof. It is enough to check for a single functional dependency $X \to Y$ whether whether the procedure decides correctly if it is in G^+. When an attribute is added to Z in lines 3–7, then Functional dependencies from G are used, thus by the soundness of the Armstrong-axioms if PRESERVE(ρ, F) claims that $X \to Y \in G^+$, then it is indeed so.

On the other hand, if $X \to Y \in G^+$, then LINEAR-CLOSURE(R, F, X) (run by G as input) adds the attributes of Y one-by-one to X. In every step when an attribute is added, some functional dependency $U \to V$ of G is used. This dependency is in one of $\pi_{R_i}(F)$'s, since G is the union of these. An easy induction on the number of functional dependencies used in procedure LINEAR-CLOSURE(R, F, X) shows that sooner or later Z becomes a subset of U, then applying the R_i-operation all attributes of V are added to Z. ■

12.3.4. Normal forms

The goal of transforming (decomposing) relational schemata into ***normal forms*** is to avoid the anomalies described in the previous section. Normal forms of many different strengths were introduced in the course of evolution of database theory, here only the ***Boyce–Codd*** normal formát (BCNF) and the ***third,*** furthermore ***fourth*** normal form (3NF and 4NF) are treated in detail, since these are the most important ones from practical point of view.

Boyce–Codd normal form

Definition 12.17 *Let R be relational schema, F be a family of functional dependencies over R. (R, F) is said to be in* ***Boyce–Codd normal form*** *if $X \to A \in F^+$ and $A \notin X$ implies that A is a superkey.*

The most important property of BCNF is that it eliminates redundancy. This is based on the following theorem whose proof is left to the Reader as an exercise (Exercise 12.3-8.).

Theorem 12.18 *Schema (R, F) is in BCNF iff for arbitrary attribute $A \in R$ and key $X \subset R$ there exists no $Y \subseteq R$, for which $X \to Y \in F^+$; $Y \to X \notin F^+$; $Y \to A \in F^+$ and $A \notin Y$.*

In other words, Theorem 12.18 states that "BCNF $\Longleftrightarrow$ There is no transitive dependence on keys". Let us assume that a given schema is not in BCNF, for example $C \to B$ and $B \to A$ hold, but $B \to C$ does not, then the same B value could occur besides many different C values, but at each occasion the same A value would be stored with it, which is redundant. Formulating somewhat differently, the meaning of BCNF is that (only) using functional dependencies an attribute value in a row cannot be predicted from other attribute

values. Indeed, assume that there exists a schema R, in which the value of an attribute can be determined using a functional dependency by comparison of two rows. That is, there exists two rows that agree on an attribute set X, differ on the set Y and the value of the remaining (unique) attribute A can be determined in one of the rows from the value taken in the other row.

X	Y	A
x	y_1	a
x	y_2	?

If the value ? can be determined by a functional dependency, then this value can only be a, the dependency is $Z \to A$, where Z is an appropriate subset of X. However, Z cannot be a superkey, since the two rows are distinct, thus R is not in BCNF.

3NF

Although BCNF helps eliminating anomalies, it is not true that every schema can be decomposed into subschemata in BCNF so that the decomposition is dependency preserving. As it was shown in Example 12.6., no proper decomposition of schema CSZ preserves the $CS \to Z$ dependency. At the same time, the schema is clearly not in BCNF, because of the $Z \to C$ dependency.

Since dependency preserving is important because of consistency checking of a database, it is practical to introduce a normal form that every schema has dependency preserving decomposition into that form, and it allows minimum possible redundancy. An attribute is called ***prime attribute,*** if it occurs in a key.

Definition 12.19 *The schema (R, F) is in **third normal form,** if whenever $X \to A \in F^+$, then either X is a superkey, or A is a prime attribute.*

The schema $SAIP$ of Example 12.3. with the dependencies $SI \to P$ and $S \to A$ is not in 3NF, since SI is the only key and so A is not a prime attribute. Thus, functional dependency $S \to A$ violates the 3NF property.

3NF is clearly weaker condition than BCNF, since "or A is a prime attribute" occurs in the definition. The schema CSZ in Example 12.6. is trivially in 3NF, because every attribute is prime, but it was already shown that it is not in BCNF.

Testing normal forms

Theoretically every functional dependency in F^+ should be checked whether it violates the conditions of BCNF or 3NF, and it is known that F^+ can be exponentially large in the size of F. Nevertheless, it can be shown that if the functional dependencies in F are of the form that the right hand side is a single attribute always, then it is enough to check violation of BCNF, or 3NF respectively, for dependencies of F. Indeed, let $X \to A \in F^+$ be a dependency that violates the appropriate condition, that is X is not a superkey and in case of 3NF, A is not prime. $X \to A \in F^+ \iff A \in X^+$. In the step when Closure(R,F,X) puts A into $X+$ (line 8) it uses a functional dependency $Y \to A$ from F that $Y \subset X^+$ and $A \notin Y$. This dependency is non-trivial and A is (still) not prime. Furthermore, if Y were a superkey, than by $R = Y^+ \subseteq (X^+)^+ = X^+$, X would also be a superkey. Thus, the functional dependency $Y \to A$ from F violates the condition of the normal form. The functional dependencies easily can be checked in polynomial time, since it is enough to calculate the closure of the left hand side of each dependency. This finishes checking for BCNF, because if the closure

of each left hand side is R, then the schema is in BCNF, otherwise a dependency is found that violates the condition. In order to test 3NF it may be necessary to decide about an attribute whether it is prime or not. However this problem is NP-complete, see Problem *12-4.*.

Lossless join decomposition into BCNF
Let (R, F) be a relational schema (where F is the set of functional dependencies). The schema is to be decomposed into union of subschemata $R_1, R_2, \ldots, R_k$, such that the decomposition has the lossless join property, furthermore each R_i endowed with the set of functional dependencies $\pi_{R_i}(F)$ is in BCNF. The basic idea of the decomposition is simple:

- If (R, F) is in BCNF, then ready.
- If not, it is decomposed into two proper parts (R_1, R_2), whose join is lossless.
- Repeat the above for R_1 and R_2.

In order to see that this works one has to show two things:

- If (R, F) is not in BCNF, then it has a lossless join decomposition into smaller parts.
- If a part of a lossless join decomposition is further decomposed, then the new decomposition has the lossless join property, as well.

Lemma 12.20 *Let (R, F) be a relational schema (where F is the set of functional dependencies), $\rho = (R_1, R_2, \ldots, R_k)$ be a lossless join decomposition of R. Furthermore, let $\sigma = (S_1, S_2)$ be a lossless join decomposition of R_1 with respect to $\pi_{R_1}(F)$. Then $(S_1, S_2, R_2, \ldots, R_k)$ is a lossless join decomposition of R.*

The proof of Lemma 12.20 is based on the associativity of natural join. The details are left to the Reader (Exercise 12.3-9.).

This can be applied for a simple, but unfortunately exponential time algorithm that decomposes a schema into subschemata of BCNF property. The projections in lines 4–5 of Naïv-BCNF(S, G) may be of exponential size in the length of the input. In order to decompose schema (R, F), the procedure must be called with parameters R, F. Procedure Naïv-BCNF(S, G) is recursive, S is the actual schema with set of functional dependencies G. It is assumed that the dependencies in G are of the form $X \to A$, where A is a single attribute.

Naïv-BCNF(S, G)

```
1 while there exists {X → A} ∈ G, that violates BCNF
2       do S_1 ← {XA}
3          S_2 ← S − A
4          G_1 ← π_{S_1}(G)
5          G_2 ← π_{S_2}(G)
6          return (Naïv-BCNF(S_1, G_1), Naïv-BCNF(S_2, G_2))
7 return S
```

However, if the algorithm is allowed overdoing things, that is to decompose a schema even if it is already in BCNF, then there is no need for projecting the dependencies. The procedure is based on the following two lemmae.

Lemma 12.21

1. *A schema of only two attributes is in BCNF.*

2. *If R is not in BCNF, then there exists two attributes A and B in R, such that $(R-AB) \to A$ holds.*

Proof. If the schema consists of two attributes, $R = AB$, then there are at most two possible non-trivial dependencies, $A \to B$ and $B \to A$. It is clear, that if some of them holds, then the left hand side of the dependency is a key, so the dependency does not violate the BCNF property. However, if none of the two holds, then BCNF is trivially satisfied.

On the other hand, let us assume that the dependency $X \to A$ violates the BCNF property. Then there must exists an attribute $B \in R - (XA)$, since otherwise X would be a superkey. For this B, $(R - AB) \to A$ holds. ■

Let us note, that the converse of the second statement of Lemma 12.21 is not true. It may happen that a schema R is in BCNF, but there are still two attributes $\{A, B\}$ that satisfy $(R - AB) \to A$. Indeed, let $R = ABC$, $F = \{C \to A, C \to B\}$. This schema is obviously in BCNF, nevertheless $(R - AB) = C \to A$.

The main contribution of Lemma 12.21 is that the projections of functional dependencies need not be calculated in order to check whether a schema obtained during the procedure is in BCNF. It is enough to calculate $(R-AB)^+$ for pairs $\{A, B\}$ of attributes, which can be done by LINEAR-CLOSURE(R, F, X) in linear time, so the whole checking is polynomial (cubic) time. However, this requires a way of calculating $(R-AB)^+$ without actually projecting down the dependencies. The next lemma is useful for this task.

Lemma 12.22 *Let $R_2 \subset R_1 \subset R$ and let F be the set of functional dependencies of scheme R. Then*

$$\pi_{R_2}(\pi_{R_1}(F)) = \pi_{R_2}(F).$$

The proof is left for the Reader (Exercise 12.3-10.). The method of lossless join BCNF decomposition is as follows. Schema R is decomposed into two subschemata. One is XA that is in BCNF, satisfying $X \to A$. The other subschema is $R - A$, hence by Theorem 12.14 the decomposition has the lossless join property. This is applied recursively to $R - A$, until such a schema is obtained that satisfies property 2 of Lemma 12.21. The lossless join property of this recursively generated decomposition is guaranteed by Lemma 12.20.

POLYNOMIAL-BCNF(R, F)

```
1  Z ← R
2  ▷ Z is the schema that is not known to be in BCNF during the procedure.
3  ρ ← ∅
4  while there exist A, B in Z, such that A ∈ (Z − AB)^+ and |Z| > 2
5        do Let A and B be such a pair
6           E ← A
7           Y ← Z − B
8           while there exist C, D in Y, such that C ∈ (Z − CD)^+
9                 do Y ← Y − D
10                   E ← C
11          ρ ← ρ ∪ {Y}
12          Z ← Z − E
13 ρ ← ρ ∪ {Z}
14 return ρ
```

The running time of POLYNOMIAL-BCNF(R, F) is polynomial, in fact it can be bounded by $O(n^5)$, as follows. During each execution of the loop in lines 4–12 the size of Z is decreased by at least one, so the loop body is executed at most n times. $(Z - AB)^+$ is calculated in line 4 for at most $O(n^2)$ pairs that can be done in linear time using LINEAR-CLOSURE that results in $O(n^3)$ steps for each execution of the loop body. In lines 8–10 the size of Y is decreased in each iteration, so during each execution of lines 3–12, they give at most n iteration. The condition of the command **while** of line 8 is checked for $O(n^2)$ pairs of attributes, each checking is done in linear time. The running time of the algorithm is dominated by the time required by lines 8–10 that take $n \cdot n \cdot O(n^2) \cdot O(n) = O(n^5)$ steps altogether.

Dependency preserving decomposition into 3NF

We have seen already that its is not always possible to decompose a schema into subschemata in BCNF so that the decomposition is dependency preserving. Nevertheless, if only 3NF is required then a decomposition can be given using MINIMAL-COVER(R, F). Let R be a relational schema and F be the set of functional dependencies. Using MINIMAL-COVER(R, F) a minimal cover G of F is constructed. Let $G = \{X_1 \to A_1, X_2 \to A_2, \ldots, X_k \to A_k\}$.

Theorem 12.23 *The decomposition $\rho = (X_1A_1, X_2A_2, \ldots, X_kA_k)$ is dependency preserving decomposition of R into subschemata in 3NF.*

Proof. Since $G^+ = F^+$ and the functional dependency $X_i \to A_i$ is in $\pi_{R_i}(F)$, the decomposition preserves every dependency of F. Let us suppose indirectly, that the schema $R_i = X_iA_i$ is not in 3NF, that is there exists a dependency $U \to B$ that violates the conditions of 3NF. This means that the dependency is non-trivial and U is not a superkey in R_i and B is not a prime attribute of R_i. There are two cases possible. If $B = A_i$, then using that U is not a superkey $U \subsetneq X_i$ follows. In this case the functional dependency $U \to A_i$ contradicts to that $X_i \to A_i$ was a member of minimal cover, since its left hand side could be decreased. In the case when $B \neq A_i$, $B \in X_i$ holds. B is not prime in R_i, thus X_i is not a key, only a superkey. However, then X_i would contain a key Y such that $Y \subsetneq X_i$. Furthermore, $Y \to A_i$ would hold, as well, that contradicts to the minimality of G since the left hand side of $X_i \to A_i$ could be decreased. ■ If the decomposition needs to have the

lossless join property besides being dependency preserving, then ρ given in Theorem 12.23 is to be extended by a key X of R. Although it was seen before that it is not possible to list ***all*** keys in polynomial time, ***one*** can be obtained in a simple greedy way, the details are left to the Reader (Exercise 12.3-11.).

Theorem 12.24 *Let (R, F) be a relational schema, and let $G = \{X_1 \to A_1, X_2 \to A_2, \ldots, X_k \to A_k\}$ be a minimal cover of F. Furthermore, let X be a key in (R, F). Then the decomposition $\tau = (X, X_1A_1, X_2A_2, \ldots, X_kA_k)$ is a lossless join and dependency preserving decomposition of R into subschemata in 3NF.*

Proof. It was shown during the proof of Theorem 12.23 that the subschemata $R_i = X_iA_i$ are in 3NF for $i = 1, 2, \ldots, k$. There cannot be a non-trivial dependency in the subschema $R_0 = X$, because if it were, then X would not be a key, only a superkey.

The lossless join property of τ is shown by the use of JOIN-TEST(R, G, ρ) procedure. Note that it is enough to consider the minimal cover G of F. More precisely, we show that the row corresponding to X in the table will be all 0 after running JOIN-TEST(R, G, ρ). Let $A_1, A_2, \ldots, A_m$ be the order of the attributes of $R - X$ as CLOSURE(R,G,X) inserts them into X^+. Since X is a key, every attribute of $R - X$ is taken during CLOSURE(R,G,X). It will be shown by induction on i that the element in row of X and column of A_i is 0 after running JOIN-TEST(R, G, ρ).

The base case of $i = 0$ is obvious. Let us suppose that the statement is true for $i-$ and consider when and why A_i is inserted into X^+. In lines 6–8 of CLOSURE(R,G,X) such a functional dependency $Y \to A_i$ is used where $Y \subseteq X \cup \{A_1, A_2, \ldots, A_{i-1}\}$. Then $Y \to A_i \in G$, $YA_i = R_j$ for some j. The rows corresponding to X and $YA_i = R_j$ agree in columns of X (all 0 by the induction hypothesis), thus the entries in column of A_i are equated by JOIN-TEST(R, G, ρ). This value is 0 in the row corresponding to $YA_i = R_j$, thus it becomes 0 in the row of X, as well. ■

It is interesting to note that although an arbitrary schema can be decomposed into subschemata in 3NF in polynomial time, nevertheless it is NP-complete to decide whether a given schema (R, F) is in 3NF, see Problem *12-4.*. However, the BCNF property can be decided in polynomial time. This difference is caused by that in order to decide 3NF property one needs to decide about an attribute whether it is prime. This latter problem requires the listing of all keys of a schema.

12.3.5. Multivalued dependencies

Example 12.8 Besides functional dependencies, some other dependencies hold in Example 12.1., as well. There can be several lectures of a subject in different times and rooms. Part of an instance of the schema could be the following.

Professor	Subject	Room	Student	Grade	Time
Caroline Doubtfire	Analysis	MA223	John Smith	A^-	Monday 8–10
Caroline Doubtfire	Analysis	CS456	John Smith	A^-	Wednesday 12–2
Caroline Doubtfire	Analysis	MA223	Ching Lee	A^+	Monday 8–10
Caroline Doubtfire	Analysis	CS456	Ching Lee	A^+	Wednesday 12–2

A set of values of Time and Room attributes, respectively, belong to each given value of Subject, and all other attribute values are repeated with these. Sets of attributes SR and StG are independent, that

is their values occur in each combination.

The set of attributes Y is said to be ***multivalued dependent*** on set of attributes X, in notation $X \twoheadrightarrow Y$, if for every value on X, there exists a set of values on Y that is not dependent in any way on the values taken in $R - X - Y$. The precise definition is as follows.

Definition 12.25 *The relational schema R satisfies the **multivalued dependency** $X \twoheadrightarrow Y$, if for every relation r of schema R and arbitrary tuples t_1, t_2 of r that satisfy $t_1[X] = t_2[X]$, there exists tuples $t_3, t_4 \in r$ such that*

- $t_3[XY] = t_1[XY]$
- $t_3[R - XY] = t_2[R - XY]$
- $t_4[XY] = t_2[XY]$
- $t_4[R - XY] = t_1[R - XY]$

holds.[1]

In Example 12.8. S↠TR holds.

Remark 12.26 *Functional dependency is **equality generating** dependency, that is from the equality of two objects it deduces the equality of other other two objects. On the other hand, multivalued dependency is **tuple generating dependency**, that is the existence of two rows that agree somewhere implies the existence of some other rows.*

There exists a sound and complete axiomatisation of multivalued dependencies similar to the Armstrong-axioms of functional dependencies. Logical implication and inference can be defined analogously. The multivalued dependency $X \twoheadrightarrow Y$ is ***logically implied*** by the set M of multivalued dependencies, in notation $M \models X \twoheadrightarrow Y$, if every relation that satisfies all dependencies of M also satisfies $X \twoheadrightarrow Y$.

Note, that $X \rightarrow Y$ implies $X \twoheadrightarrow Y$. The rows t_3 and t_4 of Definition 12.25 can be chosen as $t_3 = t_2$ and $t_4 = t_1$, respectively. Thus, functional dependencies and multivalued dependencies admit a common axiomatisation. Besides Armstrong-axioms (A1)–(A3), five other are needed. Let R be a relational schema.

(A4) ***Complementation***: $\{X \twoheadrightarrow Y\} \models X \twoheadrightarrow (R - X - Y)$.

(A5) ***Extension***: If $X \twoheadrightarrow Y$ holds, and $V \subseteq W$, then $WX \twoheadrightarrow VY$.

(A6) ***Transitivity***: $\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow (Z - Y)$.

(A7) $\{X \rightarrow Y\} \models X \twoheadrightarrow Y$.

(A8) If $X \twoheadrightarrow Y$ holds, $Z \subseteq Y$, furthermore for some W disjoint from Y $W \rightarrow Z$ holds, then $X \rightarrow Z$ is true, as well.

Beeri, Fagin and Howard proved that (A1)–(A8) is sound and complete system of axioms for functional and Multivalued dependencies together. Proof of soundness is left for the Reader (Exercise 12.3-12.), the proof of the completeness exceeds the level of this book. The rules of Lemma 12.2 are valid in exactly the same way as when only functional dependencies were considered. Some further rules are listed in the next Proposition.

[1] It would be enough to require the existence of t_3, since the existence of t_4 would follow. However, the symmetry of multivalued dependency is more apparent in this way.

Proposition 12.27 *The followings are true for multivalued dependencies.*

1. ***Union rule:*** $\{X \twoheadrightarrow Y, X \twoheadrightarrow Z\} \models X \twoheadrightarrow YZ$.
2. ***Pseudotransitivity:*** $\{X \twoheadrightarrow Y, WY \twoheadrightarrow Z\} \models WX \twoheadrightarrow (Z - WY)$.
3. ***Mixed pseudotransitivity:*** $\{X \twoheadrightarrow Y, XY \rightarrow Z\} \models X \rightarrow (Z - Y)$.
4. ***Decomposition rule*** *for multivalued dependencies: ha* $X \twoheadrightarrow Y$ *and* $X \twoheadrightarrow Z$ *holds, then* $X \twoheadrightarrow (Y \cap Z)$, $X \twoheadrightarrow (Y - Z)$ *and* $X \twoheadrightarrow (Z - Y)$ *holds, as well.*

Th proof of Proposition 12.27 is left for the Reader (Exercise 12.3-13.).

Dependency basis

Important difference between functional dependencies and multivalued dependencies is that $X \rightarrow Y$ immediately implies $X \rightarrow A$ for all A in Y, however $X \twoheadrightarrow A$ is deduced by the decomposition rule for multivalued dependencies from $X \twoheadrightarrow Y$ only if there exists a set of attributes Z such that $X \twoheadrightarrow Z$ and $Z \cap Y = A$, or $Y - Z = A$. Nevertheless, the following theorem is true.

Theorem 12.28 *Let R be a relational schema, $X \subset R$ be a set of attributes. Then there exists a partition $Y_1, Y_2, \ldots, Y_k$ of the set of attributes $R - X$ such that for $Z \subseteq R - X$ the multivalued dependency $X \twoheadrightarrow Z$ holds if and only if Z is the union of some Y_i's.*

Proof. We start from the one-element partition $W_1 = R - X$. This will be refined successively, while the property that $X \twoheadrightarrow W_i$ holds for all W_i in the actual decomposition, is kept. If $X \twoheadrightarrow Z$ and Z is not a union of some of the W_i's, then replace every W_i such that neither $W_i \cap Z$ nor $W_i - Z$ is empty by $W_i \cap Z$ and $W_i - Z$. According to the decomposition rule of Proposition 12.27, both $X \twoheadrightarrow (W_i \cap Z)$ and $X \twoheadrightarrow (W_i - Z)$ hold. Since $R - X$ is finite, the refinement process terminates after a finite number of steps, that is for all Z such that $X \twoheadrightarrow Z$ holds, Z is the union of some blocks of the partition. In order to complete the proof one needs to observe only that by the union rule of Proposition 12.27, the union of some blocks of the partition depends on X in multivalued way. ■

Definition 12.29 *The partition $Y_1, Y_2, \ldots, Y_k$ constructed in Theorem 12.28 from a set D of functional and multivalued dependencies is called the* ***dependency basis*** *of X (with respect to D).*

Example 12.9 Consider the familiar schema

R(**P**rofessor,**Su**bject,**R**oom,**St**udent,**G**rade,**T**ime)

of Examples 12.1. and 12.8.. **Su→→RT** was shown in Example 12.8.. By the complementation rule **Su→→PStG** follows. **Su→P** is also known. This implies by axiom (A7) that **Su→→P**. By the decomposition rule **Su→→Stg** follows. It is easy to see that no other one-element attribute set is determined by **Su** via multivalued dependency. Thus, the dependency basis of **Su** is the partition{**P**,**RT**,**StG**}.

We would like to compute the set D^+ of logical consequences of a given set D of functional and multivalued dependencies. One possibility is to apply axioms (A1)–(A8) to extend the set of dependencies repeatedly, until no more extension is possible. However, this could be an exponential time process in the size of D. One cannot expect any better, since it

was shown before that even D^+ can be exponentially larger than D. Nevertheless, in many applications it is not needed to compute the whole set D^+, one only needs to decide whether a given functional dependency $X \to Y$ or multivalued dependency $X \twoheadrightarrow Y$ belongs to D^+ or not. In order to decide about a multivalued dependency $X \twoheadrightarrow Y$, it is enough to compute the dependency basis of X, then to check whether $Z - X$ can be written as a union of some blocks of the partition. The following is true.

Theorem 12.30 (Beeri). *In order to compute the dependency basis of a set of attributes X with respect to a set of dependencies D, it is enough to consider the following set M of multivalued dependencies:*

1. *All multivalued dependencies of D and*
2. *for every $X \to Y$ in D the set of multivalued dependencies $X \twoheadrightarrow A_1, X \twoheadrightarrow A_2, \ldots, X \twoheadrightarrow A_k$, where $Y = A_1 A_2 \ldots A_k$, and the A_i's are single attributes.*

The only thing left is to decide about functional dependencies based on the dependency basis. Closure(R,F,X) works correctly only if multivalued dependencies are not considered. The next theorem helps in this case.

Theorem 12.31 (Beeri). *Let us assume that $A \notin X$ and the dependency basis of X with respect to the set M of multivalued dependencies obtained in Theorem 12.30 is known. $X \to A$ holds if and only if*

1. *A forms a single element block in the partition of the dependency basis, and*
2. *There exists a set Y of attributes that does not contain A, $Y \to Z$ is an element of the originally given set of dependencies D, furthermore $A \in Z$.*

Based on the observations above, the following polynomial time algorithm can be given to compute the dependency basis of a set of attributes X.

Dependency-basis(R, M, X)

```
1 S ← {R − X}                          ▷ The collection of sets in the dependency basis is S.
2 repeat
3       for all V ↠ W ∈ M
4           do if there exists Y ∈ S such that Y ∩ W ≠ ∅ ∧ Y ∩ V = ∅
5                 then S ← S − {{Y}} ∪ {{Y ∩ W}, {Y − W}}
6 until S does not change
7 return S
```

It is immediate that if $\mathcal{S}$ changes in lines 3–5. of Dependency-basis(R, M, X), then some block of the partition is cut by the algorithm. This implies that the running time is a polynomial function of the sizes of M and R. In particular, by careful implementation one can make this polynomial to $O(|M| \cdot |R|^3)$, see Problem 12-5..

Fourth normal form 4NF

The Boyce–Codd normal form can be generalised to the case where multivalued dependencies are also considered besides functional dependencies, and one needs to get rid of the redundancy caused by them.

Definition 12.32 *Let R be a relational schema, D be a set of functional and multivalued dependencies over R. R is in **fourth normal form (4NF)**, if for arbitrary multivalued dependency $X \twoheadrightarrow Y \in D^+$ for which $Y \not\subseteq X$ and $R \neq XY$, holds that X is superkey in R.*

Observe that 4NF$\Longrightarrow$BCNF. Indeed, if $X \to A$ violated the BCNF condition, then $A \notin X$, furthermore XA could not contain all attributes of R, because that would imply that X is a superkey. However, $X \to A$ implies $X \twoheadrightarrow A$ by (A8), which in turn would violate the 4NF condition.

Schema R together with set of functional and multivalued dependencies D can be decomposed into $\rho = (R_1, R_2, \ldots, R_k)$, where each R_i is in 4NF and the decomposition has the lossless join property. The method follows the same idea as the decomposition into BCNF subschemata. If schema S is not in 4NF, then there exists a multivalued dependency $X \twoheadrightarrow Y$ in the projection of D onto S that violates the 4NF condition. That is, X is not a superkey in S, Y neither is empty, nor is a subset of X, furthermore the union of X and Y is not S. It can be assumed without loss of generality that X and Y are disjoint, since $X \twoheadrightarrow (Y - X)$ is implied by $X \twoheadrightarrow Y$ using (A1), (A7) and the decomposition rule. In this case S can be replaced by subschemata $S_1 = XY$ and $S_2 = S - Y$, each having a smaller number of attributes than S itself, thus the process terminates in finite time.

Two things has to be dealt with in order to see that the process above is correct.

- Decomposition S_1, S_2 has the lossless join property.
- How can the projected dependency set $\pi_S(D)$ be computed?

The first problem is answered by the following theorem.

Theorem 12.33 *The decomposition $\rho = (R_1, R_2)$ of schema R has the lossless join property with respect to a set of functional and multivalued dependencies D iff*

$$(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2).$$

Proof. The decomposition $\rho = (R_1, R_2)$ of schema R has the lossless join property iff for any relation r over the schema R that satisfies all dependencies from D holds that if μ and ν are two tuples of r, then there exists a tuple φ satisfying $\varphi[R_1] = \mu[R_1]$ and $\varphi[R_2] = \nu[R_2]$, then it is contained in r. More precisely, φ is the natural join of the projections of μ on R_1 and of ν on R_2, respectively, which exist iff $\mu[R_1 \cap R_2] = \nu[R_1 \cap R_2]$. Thus the fact that φ is always contained in r is equivalent with that $(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)$. ■

To compute the projection $\pi_S(D)$ of the dependency set D one can use the following theorem of Aho, Beeri and Ullman. $\pi_S(D)$ is the set of multivalued dependencies that are logical implications of D and use attributes of S only.

Theorem 12.34 (Aho, Beeri és Ullman). *$\pi_S(D)$ consists of the following dependencies:*

- *For all $X \to Y \in D^+$, if $X \subseteq S$, then $X \to (Y \cap S) \in \pi_S(D)$.*
- *For all $X \twoheadrightarrow Y \in D^+$, if $X \subseteq S$, then $X \twoheadrightarrow (Y \cap S) \in \pi_S(D)$.*

Other dependencies cannot be derived from the fact that D holds in R.

Unfortunately this theorem does not help in computing the projected dependencies in polynomial time, since even computing D^+ could take exponential time. Thus, the algorithm of 4NF decomposition is not polynomial either, because the 4NF condition must be checked with respect to the projected dependencies in the subschemata. This is in deep contrast with

the case of BCNF decomposition. The reason is, that to check BCNF condition one does not need to compute the projected dependencies, only closures of attribute sets need to be considered according to Lemma 12.21.

Exercises

12.3-1 Are the following inference rules sound?

a. If $XW \to Y$ and $XY \to Z$, then $X \to (Z - W)$.

b. If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow Z$.

c. If $X \twoheadrightarrow Y$ and $XY \to Z$, then $X \to Z$.

12.3-2 Prove Theorem 12.30, that is show the following. Let D be a set of functional and multivalued dependencies, and let $m(D) = \{X \twoheadrightarrow Y : X \twoheadrightarrow Y \in D\} \cup \{X \twoheadrightarrow A : A \in Y \text{ for some } X \to Y \in D\}$. Then

a. $D \models X \to Y \Longrightarrow m(D) \models X \twoheadrightarrow Y$, and

b. $D \models X \twoheadrightarrow Y \iff m(D) \models X \twoheadrightarrow Y$.

Hint. Use induction on the inference rules to prove b.

12.3-3 Consider the database of an investment firm, whose attributes are as follows: B (stockbroker), O (office of stockbroker), I (investor), S (stock), A (amount of stocks of the investor), D (dividend of the stock). The following functional dependencies are valid: $S \to D$, $I \to B$, $IS \to A$, $B \to O$.

a. Determine a key of schema $R = BOISAD$.

b. How many keys are in schema R?

c. Give a lossless join decomposition of R into subschemata in BCNF.

d. Give a dependency preserving and lossless join decomposition of R into subschemata in 3NF.

12.3-4 The schema R of Exercise 12.3-3. is decomposed into subschemata SD, IB, ISA and BO. Does this decomposition have the lossless join property?

12.3-5 Assume that schema R of Exercise 12.3-3. is represented by ISA, IB, SD and ISO subschemata. Give a minimal cover of the projections of dependencies given in Exercise 12.3-3.. Exhibit a minimal cover for the union of the sets of projected dependencies. Is this decomposition dependency preserving?

12.3-6 Let the functional dependency $S \to D$ of Exercise 12.3-3. be replaced by the multivalued dependency $S \twoheadrightarrow D$. That is , D represents the stock's dividend "history".

a. Compute the dependency basis of I.

b. Compute the dependency basis of BS.

c. Give a decomposition of R into subschemata in 4NF.

12.3-7 Consider the decomposition $\rho = \{R_1, R_2, \ldots, R_k\}$ of schema R. Let $r_i = \pi_{R_i}(r)$, furthermore $m_\rho(r) = \bowtie_{i=1}^{k} \pi_{R_i}(r)$. Prove:

a. $r \subseteq m_\rho(r)$.

b. If $s = m_\rho(r)$, then $\pi_{R_i}(s) = r_i$.
c. $m_\rho(m_\rho(r)) = m_\rho(r)$.

12.3-8 Prove that schema (R, F) is in BCNF iff for arbitrary $A \in R$ and key $X \subset R$, it holds that there exists no $Y \subseteq R$, for which $X \to Y \in F^+$; $Y \to X \notin F^+$; $Y \to A \in F^+$ and $A \notin Y$.
12.3-9 Prove Lemma 12.20.
12.3-10 Let us assume that $R_2 \subset R_1 \subset R$ and the set of functional dependencies of schema R is F. Prove that $\pi_{R_2}(\pi_{R_1}(F)) = \pi_{R_2}(F)$.
12.3-11 Give a $O(n^2)$ running time algorithm to find a key of the relational schema (R, F). *Hint.* Use that R is superkey and each superkey contains a key. Try to drop attributes from R one-by-one and check whether the remaining set is still a key.
12.3-12 Prove that axioms (A1)–(A8) are sound for functional and multivalued dependencies.
12.3-13 Derive the four inference rules of Proposition 12.27 from axioms (A1)–(A8).

12.4. Generalised dependencies

Two such dependencies will be discussed in this section that are generalizations of the previous ones, however cannot be axiomatised with axioms similar to (A1)–(A8).

12.4.1. Join dependencies

Theorem 12.33 states that multivalued dependency is equivalent with that some decomposition the schema into two parts has the lossless join property. Its generalisation is the ***join dependency***.

Definition 12.35 *Let R be a relational schema and let $R = \bigcup_{i=1}^{k} X_i$. The relation r belonging to R is said to satisfy the* ***join dependency***

$$\bowtie[X_1, X_2, \ldots, X_k]$$

if

$$r = \bowtie_{i=1}^{k} \pi_{X_i}(r).$$

In this setting r satisfies multivalued dependency $X \twoheadrightarrow Y$ iff it satisfies the join dependency $\bowtie[XY, X(R-Y)]$. The join dependency $\bowtie[X_1, X_2, \ldots, X_k]$ expresses that the decomposition $\rho = (X_1, X_2, \ldots, X_k)$ has the lossless join property. One can define the ***fifth normal form, 5NF***.

Definition 12.36 *The relational schema R is in* ***fifth normal form,*** *if it is in 4NF and has no non-trivial join dependency.*

The fifth normal form has theoretical significance primarily. The schemata used in practice usually have ***primary keys***. Using that the schema could be decomposed into subschemata of two attributes each, where one of the attributes is a superkey in every subschema.

Example 12.10 Consider the database of clients of a bank (Clientnumber,Name,Address,accountBalance). Here C is unique identifier, thus the schema could be

decomposed into (CN,CA,CB), which has the lossless join property. However, it is not worth doing so, since no storage place can be saved, furthermore no anomalies are avoided with it.

There exists an ***axiomatisation*** of a dependency system if there is a finite set of inference rules that is sound and complete, i.e. logical implication coincides with being derivable by using the inference rules. For example, the Armstrong-axioms give an axiomatisation of functional dependencies, while the set of rules (A1)–(A8) is the same for functional and multivalued dependencies considered together. Unfortunately, the following negative result is true.

Theorem 12.37 *The family of join dependencies has no finite axiomatisation.*

In contrary to the above, Abiteboul, Hull and Vianu show in their book that the logical implication problem can be decided by an algorithm for the family of functional and join dependencies taken together. The complexity of the problem is as follows.

Theorem 12.38

- *It is NP-complete to decide whether a given join dependency is implied by another given join dependency and a functional dependency.*
- *It is NP-hard to decide whether a given join dependency is implied by given set of multivalued dependencies.*

12.4.2. Branching dependencies

A generalisation of functional dependencies is the family of ***branching dependencies***. Let us assume that $A, B \subset R$ and there exists no $q+1$ rows in relation r over schema R, such that they contain at most p distinct values in columns of A, but all $q+1$ values are pairwise distinct in some column of B. Then B is said to be (p,q)***-dependent*** on A, in notation $A \xrightarrow{p,q} B$. In particular, $A \xrightarrow{1,1} B$ holds if and only if functional dependency $A \to B$ holds.

Example 12.11

Consider the database of the trips of an international transport truck.

- One trip: four distinct countries.
- One country has at most five neighbours.
- There are 30 countries to be considered.

Let x_1, x_2, x_3, x_4 be the attributes of the countries reached in a trip. In this case $x_i \xrightarrow{1,1} x_{i+1}$ does not hold, however another dependency is valid:

$$x_i \xrightarrow{1,5} x_{i+1}.$$

The storage space requirement of the database can be significantly reduced using these dependencies. The range of each element of the original table consists of 30 values, names of countries or some codes of them (5 bits each, at least). Let us store a little table ($30 \times 5 \times 5 = 750$ bits) that contains a numbering of the neighbours of each country, which assigns to them the numbers 0,1,2,3,4 in some order. Now we can replace attribute x_2 by these numbers (x_2^*), because the value of x_1 gives the starting country and the value of x_2^* determines the second country with the help of the little table. The same holds for the attribute x_3, but we can decrease the number of possible values even further, if we give

a table of numbering the possible third countries for each x_1, x_2 pair. In this case, the attribute x_3^* can take only 4 different values. The same holds for x_4, too. That is, while each element of the original table could be encoded by 5 bits, now for the cost of two little auxiliary tables we could decrease the length of the elements in the second column to 3 bits, and that of the elements in the third and fourth columns to 2 bits.

The (p, q)-closure of an attribute set $X \subset R$ can be defined:

$$C_{p,q}(X) = \{A \in R \colon X \xrightarrow{p,q} A\}.$$

In particular, $C_{1,1}(X) = X^+$. In case of branching dependencies even such basic questions are hard as whether there exists an ***Armstrong-relation*** for a given family of dependencies.

Definition 12.39 *Let R be a relational schema, F be a set of dependencies of some dependency family $\mathcal{F}$ defined on R. A relation r over schema R is **Armstrong-relation** for F, if the set of dependencies from $\mathcal{F}$ that r satisfies is exactly F, that is $F = \{\sigma \in \mathcal{F} \colon r \models \sigma\}$.*

Armstrong proved that for an arbitrary set of functional dependencies F there exists Armstrong-relation for F^+. The proof is based on the three properties of closures of attributes sets with respect to F, listed in Exercise 12.2-1. For branching dependencies only the first two holds in general.

Lemma 12.40 *Let $0 < p \leq q$, furthermore let R be a relational schema. For $X, Y \subseteq R$ one has*

1. $X \subseteq C_{p,q}(X)$ *and*
2. $X \subseteq Y \Longrightarrow C_{p,q}(X) \subseteq C_{p,q}(Y)$.

There exists such $C \colon 2^R \to 2^R$ mapping and natural numbers p, q that there exists no Armstrong-relation for C in the family if (p, q)-dependencies.

Grant Minker investigated ***numerical dependencies*** that are similar to branching dependencies. For attribute sets $X, Y \subseteq R$ the dependency $X \xrightarrow{k} Y$ holds in a relation r over schema R if for every tuple value taken on the set of attributes X, there exists at most k distinct tuple values taken on Y. This condition is stronger than that of $X \xrightarrow{1,k} Y$, since the latter only requires that in each column of Y there are at most k values, independently of each other. That allows $k^{|Y-X|}$ different Y projections. Numerical dependencies were axiomatised in some special cases, based on that Katona showed that branching dependencies have no finite axiomatisation. It is still an open problem whether logical implication is algorithmically decidable amongst branching dependencies.

Exercises

12.4-1 Prove Theorem 12.38.
12.4-2 Prove Lemma 12.40.
12.4-3 Prove that if $p = q$, then $C_{p,p}\left(C_{p,p}(X)\right) = C_{p,p}(X)$ holds besides the two properties of Lemma 12.40.
12.4-4 A $C \colon 2^R \to 2^R$ mapping is called a ***closure***, if it satisfies the two properties of Lemma 12.40 and and the third one of Exercise 12.4-3.. Prove that if $C \colon 2^R \to 2^R$ is a closure, and F is the family of dependencies defined by $X \to Y \iff Y \subseteq C(X)$, then there exists an Armstrong-relation for F in the family of $(1, 1)$-dependencies (functional dependencies) and in the family of $(2, 2)$-dependencies, respectively.

12.4-5 Let C be the closure defined by

$$C(X) = \begin{cases} X, & \text{if } |X| < 2 \\ R & \text{otherwise.} \end{cases}$$

Prove that there exists no Armstrong-relation for C in the family of (n, n)-dependencies, if $n > 2$.

Problems

12-1. External attributes

Maier calls attribute A an ***external attribute*** in the functional dependency $X \to Y$ with respect to the family of dependencies F over schema R, if the following two conditions hold:

1. $(F - \{X \to Y\}) \cup \{X \to (Y - A)\} \models X \to Y$, or
2. $(F - \{X \to Y\}) \cup \{(X - A) \to Y\} \models X \to Y$.

Design an $O(n^2)$ running time algorithm, whose input is schema (R, F) and output is a set of dependencies G equivalent with F that has no external attributes.

12-2. The order of the elimination steps in the construction of minimal cover is important

In the procedure MINIMAL-COVER(R, F) the set of functional dependencies was changed in two ways: either by dropping redundant dependencies, or by dropping redundant attributes from the left hand sides of the dependencies. If the latter method is used first, until there is no more attribute that can be dropped from some left hand side, then the first method, this way a minimal cover is obtained really, according to Proposition 12.6. Prove that if the first method applied first and then the second, until there is no more possible applications, respectively, then the obtained set of dependencies is not necessarily a minimal cover of F.

12-3. BCNF subschema

Prove that the following problem is coNP-complete: Given a relational schema R with set of functional dependencies F, furthermore $S \subset R$, decide whether $(S, \pi_S(F))$ is in BCNF.

12-4. 3NF is hard to recognise

Let (R, F) be a relational schema, where F is the system of functional dependencies.
The k ***size key problem*** is the following: given a natural number k, determine whether there exists a key of size at most k.
The ***prime attribute problem*** is the following: for a given $A \in R$, determine whether it is a prime attribute.

a. Prove that the k size key problem is NP-complete. *Hint.* Reduce the ***vertex cover*** problem to the prime attribute problem.

b. Prove that the prime attribute problem is NP-complete by reducing the k size key problem to it.

c. Prove that determining whether the relational schema (R, F) is in 3NF is NP-complete. *Hint.* Reduce the prime attribute problem to it.

***12-5. Running time of* DEPENDENCY-BASIS**
Give an implementation of procedure DEPENDENCY-BASIS, whose running time is $O(|M| \cdot |R|^3)$.

Chapter notes

The relational data model was introduced by Codd [9] in 1970. Functional dependencies were treated in his paper of 1972 [13], their axiomatisation was completed by Armstrong [3]. The logical implication problem for functional dependencies were investigated by Beeri and Bernstein [5], furthermore Maier [24]. Maier also treats the possible definitions of minimal covers, their connections and the complexity of their computations in that paper. Maier, Mendelzon and Sagiv found method to decide logical implications among general dependencies [25]. Beeri, Fagin and Howard proved that axiom system (A1)–(A8) is sound and complete for functional and multivalued dependencies taken together [7]. Yu and Johnson [31] constructed such relational schema, where $|F| = k$ and the number of keys is $k!$. Békéssy and Demetrovics [8] gave a simple and beautiful proof for the statement, that from k functional dependencies at most $k!$ keys can be obtained, thus Yu and Johnson's construction is extremal.

Armstrong-relations were introduced and studied by Fagin [19, 20], furthermore by Beeri, Fagin, Dowd and Statman [6].

Multivalued dependencies were independently discovered by Zaniolo [32], Fagin [18] and Delobel [14].

The necessity of normal forms was recognised by Codd while studying update anomalies [11, 12]. The Boyce–Codd normal form was introduced in [10]. The definition of the third normal form used in this chapter was given by Zaniolo [33]. Complexity of decomposition into subschemata in certain normal forms was studied by Lucchesi and Osborne [23], Beeri and Bernstein [5], furthermore Tsou and Fischer [29].

Theorems 12.30 and 12.31 are results of Beeri [4]. Theorem 12.34 is from a paper of Aho, Beeri és Ullman [2].

Theorems 12.37 and 12.38 are from the book of Abiteboul, Hull and Vianu [1], the non-existence of finite axiomatisation of join dependencies is Petrov's result [26].

Branching dependencies were introduced by Demetrovics, Katona and Sali, they studied existence of Armstrong-relations and the size of minimal Armstrong-relations [15, 16, 17, 27]. Katona showed that there exists no finite axiomatisation of branching dependencies in (ICDT'92 Berlin, invited talk) but never published.

Possibilities of axiomatisation of numerical dependencies were investigated by Grant and Minker [21, 22].

Good introduction of the concepts of this chapter can be found in the books of Abiteboul, Hull and Vianu [1], Ullman [30] furthermore Thalheim [28], respectively.

Bibliography

[1] S. Abiteboul, V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995. 521

[2] A. Aho, C. Beeri, J. D. Ullman. The theory of joins in relational databases. *ACM Transactions on Database Systems*, 4(3):297–314, 1979. 521

[3] W. Armstrong. Dependency structures of database relationships. In *Proceedings of IFIP Congress*, 580–583. o. North Holland, 1974. 521

[4] C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Transactions on Database Systems*, 5:241–259, 1980. 521

[5] C. Beeri, P. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems*, 4(1):30–59, 1979. 521

[6] C. Beeri, M. Dowd. On the structure of armstrong relations for functional dependencies. *Journal of ACM*, 31(1):30–46, 1984. 521

[7] C. Beeri, R. Fagin, J. Howard. A complete axiomatization for functional and multivalued dependencies. In *ACM SIGMOD Symposium on the Management of Data*, 47–61. o., 1977. 521

[8] A. Békéssy, J. Demetrovics. Contribution to the theory of data base relations. *Discrete Mathematics*, 27(1):1–10, 1979. 521

[9] E. F. Codd. A relational model of large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. 521

[10] E. F. Codd. Recent investigations in relational data base systems. In *Information Processing 74*. North-Holland, pp. 1017–1021,1974. 521

[11] E. F. Codd. Normalized database structure: A brief tutorial. In *ACM SIGFIDET Workshop on Data Description, Access and Control*, pp. 24–30, 1971. 521

[12] E. F. Codd. Further normalization of the data base relational model. In R. Rustin (szerkesztő), *Courant Computer Science Symposium 6: Data Base Systems*. Prentice Hall, pp. 33–64, 1972. 521

[13] E. F. Codd. Relational completeness of database sublanguages. In *Courant Computer Science Symposium 6: Data Base Systems*. Prentice Hall, pp. 65–98, 1972, editor =. 521

[14] C. Delobel. Normalization and hierarchical dependencies in the relational data model. *ACM Transactions on Database Systems*, 3(3):201–222, 1978. 521

[15] J. Demetrovics, Gy. O. H. Katona, A. Sali. Minimal representations of branching dependencies. *Discrete Applied Mathematics*, 40:139–153, 1992. 521

[16] J. Demetrovics, Gy. O. H. Katona, A. Sali. Minimal representations of branching dependencies. *Acta Scientiarum Mathematicorum (Szeged)*, 60:213–223, 1995. 521

[17] J. Demetrovics, Gy. O. H. Katona, A. Sali. Design type problems motivated by database theory. *Journal of Statistical Planning and Inference*, 72:149–164, 1998. 521

[18] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems*, 2:262–278, 1977. 521

[19] R. Fagin. Armstrong databases. In *Proceedings of IBM Symposium on Mathematical Foundations of Computer Science*, 1982. 521

[20] R. Fagin. Horn clauses and database dependencies. *Journal of ACM*, 29(4):952–985, 1982. 521

[21] J. Grant, J. Minker. Inferences for numerical dependencies. *Theoretical Computer Science*, 41:271–287, 1985. 521

[22] J. Grant, J. Minker. Normalization and axiomatization for numerical dependencies. *Information and Control*, 65:1–17, 1985. 521

[23] C. Lucchesi. Candidate keys for relations. *Journal of of Computer and System Sciences*, 17(2):270–279, 1978. 521

[24] D. Maier. Minimum covers in the relational database model. *Journal of the ACM*, 27(4):664–674, 1980. 521

[25] D. Maier, A. O. Mendelzon, Y. Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4):455–469, 1979. 521

[26] S. Petrov. Finite axiomatization of languages for representation of system properties. *Information Sciences*, 47:339–372, 1989. 521

[27] A. Sali, Sr., A. Sali. Generalized dependencies in relational databases. *Acta Cybernetica*, 13:431–438, 1998. 521

[28] B. Thalheim. *Dependencies in Relational Databases*. B. G. Teubner, 1991. 521

[29] D. M. Tsou, P. C. Fischer. Decomposition of a relation scheme into Boyce–Codd normal form. *SIGACT News*, 14(3):23–29, 1982. 521

[30] J. D. Ullman. *Principles of Database and Knowledge Base Systems. Vol. 1.* Computer Science Press, 1989 (2. edition). 521

[31] C. Yu, D. Johnson. On the complexity of finding the set of candidate keys for a given set of functional dependencies. In *Information Processing 74*. North-Holland, pp. 580–583, 1974. 521

[32] C. Zaniolo. Analysis and design of relational schemata for database systems. Technical Report UCLA–Eng–7669, Department of Computer Science, University of California at Los Angeles, 1976. 521

[33] C. Zaniolo. A new normal form for the design of relational database schemata. *ACM Transactions on Database Systems*, 7:489–499, 1982. 521

Name index

Subject Index

Contents