Information Systems

Chapter 2:

Relational Database Theory

Rita Schindler | TU Ilmenau, Germany
www.tu-ilmenau.de/dbis

Relational Model: Overview

- Relational Model was introduced in 1970 by E.F. Codd (at IBM).
- Nice features:
  - Simple and uniform data structures – relations – and
  - solid theoretical foundation (important for query processing and optimization)
- Relational Model is basis for most DBMS, e.g.,
  - Oracle, Microsoft SQL Server, IBM DB2, Sybase, Postgres, MySQL, . . .
- Typically used in conceptual design:
  - either directly (creating tables using SQL DDL) or
  - derived from a given Entity-Relationship schema.

Basic Structure of the Relational Model

- A relation r over a collection of sets (domain values) $D_1, D_2, \ldots, D_n$ is a subset of the Cartesian Product $D_1 \times D_2 \times \ldots \times D_n$
- A relation thus is a set of n-tuples $(d_1, d_2, \ldots, d_n)$, where $d_i \in D_i$.

Given the sets

$StudId$ = { 412, 307, 540 },

$StudName$ = { Smith, Jones },

$Major$ = { CS, CSE, BIO }

then

r = {(412, Smith, CS), (307, Jones, CSE), (412, Smith, CSE)}

is a relation over $StudId \times StudName \times Major$

Relation Schema, Database Schema, and Instances

- Let A_1, A_2, …, A_n be attributes with domains D_1, D_2, …, D_n,
  then $R(A_1: D_1, A_2: D_2, \ldots, A_n: D_n)$ is a relation schema.
  - Example:
    Student(StudId: number, StudName: string, Major: string)
- A relation schema specifies the name and the structure of the relation.
- A collection of relation schemas is called a relational database schema.

Relation Schema, Database Schema, and Instances

- A relation instance $r(R)$ of a relation schema can be thought of as a table with n columns and a number of rows.
- Instead of relation instance we often just say relation.
- An instance of a database schema thus is a collection of relations.
- An element $t \in r(R)$ is called a tuple (or row).

Student

StudId	StudName	Major	
			← relation schema
123	Smith	CS	
235	Jones	CSE	← tuple
367	Miller	CSE	

- A relation has the following properties:
  - the order of rows is irrelevant, and
  - there are no duplicate rows in a relation

Integrity Constraints in the Relational Model

- Integrity constraints (ICs): must be true for any instance of a relation schema (admissible instances)
  - ICs are specified when the schema is defined
  - ICs are checked by the DBMS when relations (instances) are modified
- If DBMS checks ICs, then the data managed by the DBMS more closely correspond to the real-world scenario that is being modeled!

Primary Key Constraints

- A set of attributes is a key for a relation if:
  1. no two distinct tuples have the same values for all key attributes, and
  2. this is not true for any subset of that key.

- If there is more than one key for a relation (i.e., we have a set of candidate keys), one is chosen (by the designer or DBA) to be the primary key.
  - Example:
    Student(<u>StudId: number</u>, StudName: string, Major: string)
  - For candidate keys not chosen as primary key, uniqueness constraints can be specified.
  - Note that it is often useful to introduce an artificial primary key (as a single attribute) for a relation, in particular if this relation is often "referenced".

Foreign Key Constraints and Referential Integrity

- Set of attributes in one relation (child relation) that is used to "refer" to a tuple in another relation (parent relation). Foreign key must refer to the primary key of the referenced relation.
- Foreign key attributes are required in relation schemas that have been derived from relationship types.
  - Example:
    offers(Prodname $\rightarrow$ PRODUCTS, SName $\rightarrow$ SUPPLIERS, Price)
    orders((FName, LName) $\rightarrow$ CUSTOMERS, Quantity,
    SName $\rightarrow$ SUPPLIERS, Prodname $\rightarrow$ PRODUCTS)
- Foreign/primary key attributes must have matching domains.
- A foreign key constraint is satisfied for a tuple if
  - ***either*** some values of the foreign key attributes are null (meaning a reference is not known),
  - ***or*** the values of the foreign key attributes occur as the values of the primary key (of some tuple) in the parent relation.

Foreign Key Constraints and Referential Integrity

- The combination of foreign key attributes in a relation schema typically builds the primary key of the relation.
  - Example:
    M:N - relationship
    offers (<u>Prodname → PRODUCTS, SName → SUPPLIERS</u>, Price)
- If all foreign key constraints are enforced for a relation, referential integrity is achieved, i.e., there are no dangling references.

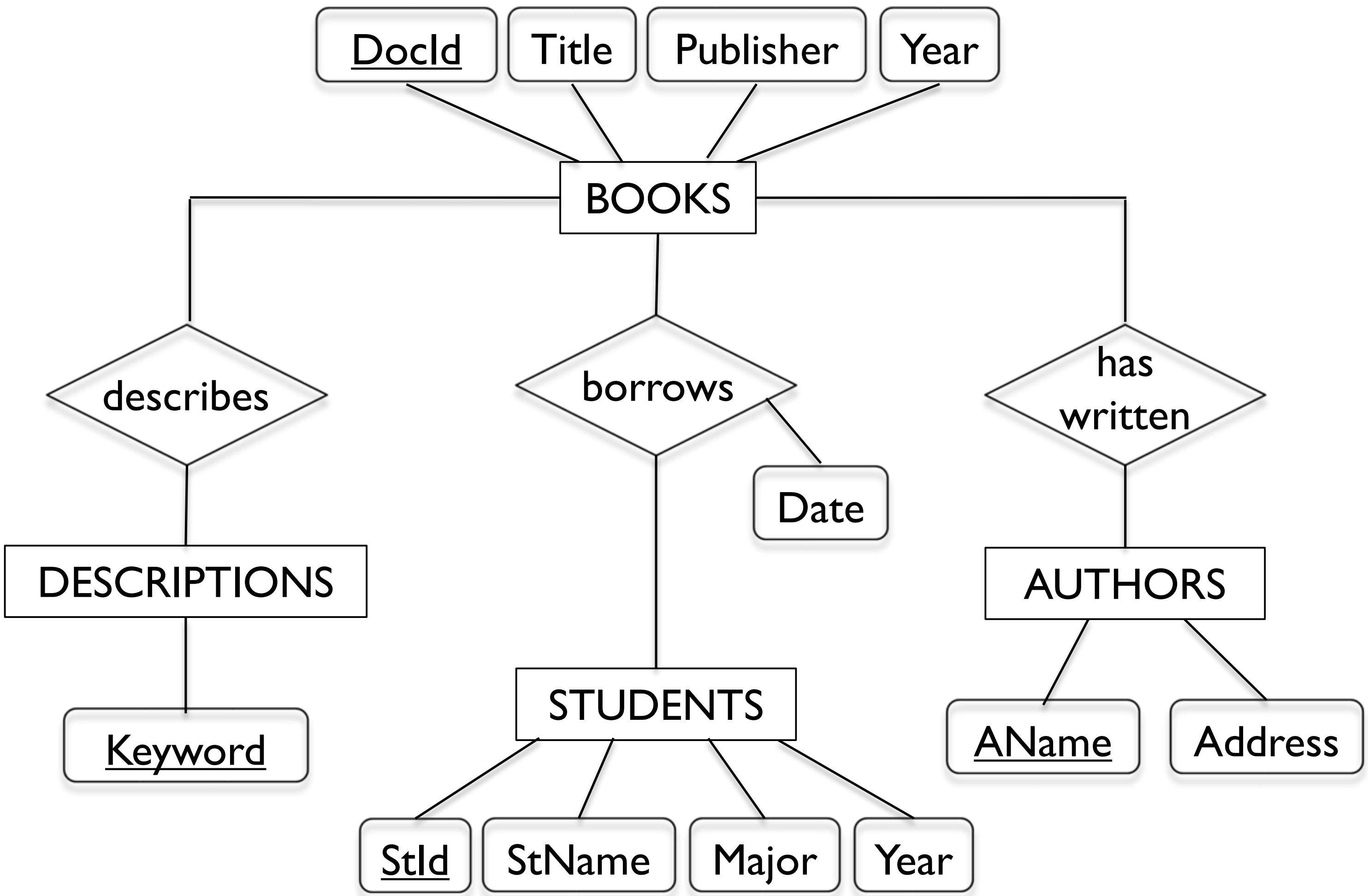
Translation of an ER Schema into a Relational Schema

- Entity type $E(A_1, \ldots, A_n, B_1, \ldots, B_m)$
  $\Rightarrow$ relation schema $E(A_1, \ldots, A_n, B_1, \ldots, B_m)$.

- Relationship type $R(E_1,\ldots,E_n,A_1,\ldots,A_m)$ with participating entity types $E_1, \ldots, E_n$;
  X_i ≡ foreign key attribute(s) referencing primary key attribute(s) of relation schema corresponding to E_i.
  $\Rightarrow R(X_1 \rightarrow E_1, \ldots, X_n \rightarrow E_n, A_1, \ldots, A_m)$

- For a functional relationship (N:1, 1:N), an optimization is possible. Assume N:1 relationship type between E_1 and E_2. We can extend the schema of E_1 to
  $E_1(A_1,\ldots,A_n, X_2 \rightarrow E_2, B_1,\ldots,B_m)$,
  - e.g., EMPLOYEES(EmpId, DeptNo → DEPARTMENTS, ...)

Example


DocId
Title
Publisher
Year
BOOKS
describes
borrows
has written
Date
DESCRIPTIONS
AUTHORS
Keyword
STUDENTS
AName
Address
StId
StName
Major
Year

Example

- According to step 1:
  BOOKS(DocId, Title, Publisher, Year)
  STUDENTS(StId, StName, Major, Year)
  DESCRIPTIONS(Keyword)
  AUTHORS(AName, Address)
- In step 2 the relationship types are translated:
  borrows(DocId → BOOKS, StId → STUDENTS, Date)
  has-written(DocId → BOOKS, AName → AUTHORS)
  describes(DocId → BOOKS, Keyword → DESCRIPTIONS)
- No need for extra relation for entity type DESCRIPTIONS:
  describes(DocId → BOOKS, Keyword)

Relational Database Design

- Refinement of logical design step
- **Goal**: avoid redundancies by splitting relational schemas without
  - losing semantic information (dependency preserving decomposition)
  - opportunity to reconstruct original relation (lossless decomposition)
- avoid redundancies by transformating schemas into normal forms

Example: Relation Containing Redundancies (1)

CNo	CName	Flight	Destination	Country	Airline
K1013	Meier, R.	M107	Paris	F	Sea Gull
K1013	Meier, R.	AT286	Edmonton	CA	AirTrans
K8516	Schulz, B.	AT286	Edmonton	CA	AirTrans
K1005	Koch, A.u.E.	A456	Paris	F	Albatross
K5313	Walter, S.	M117	London	GB	Sea Gull
K1013	Meier, R.	A432	Vancouver	CA	Albatross

- Problems:
  - Insertion of a new booking for flight A456 or insertion for a new flight without bookings
  - Flight A456 is cancelled
  - Update of the name from customer 'Meier, R.'

Example: Relation Containing Redundancies (2)

Customers

Flights

CNo	Name
K1013	Meier, R.
K1013	Meier, R.
K8516	Schulz, B.
K1005	Koch, A.u.E.
K5313	Walter, S.
K1013	Meier, R.

Flight	Destination	Country	Airline
M107	Paris	F	Sea Gull
AT286	Edmonton	CA	AirTrans
AT286	Edmonton	CA	AirTrans
A456	Paris	F	Albatross
M117	London	GB	Sea Gull
A432	Vancouver	CA	Albatross

Relation Containing Redundancies

StudentID	Name	Course	Grade	Program	Faculty
1234	Bob	Operating Systems	1.7	CS	Comp. Science
1235	Peter	Database Systems	2.3	CS	Comp. Science
1236	Mary	Analysis	1.0	EE	Electr. Eng.
1239	Steve	Database Systems	1.3	CS	Comp. Science

Redundancies

- Avoid redundancies in base relations for different reasons:
  - redundant information require additional disk space
  - updates on base relations containing redundancies are difficult to process correctly based only on local integrity constraints: all occurences of a given information have to be updated

Update Anomalies

- Insert a record into the relation containing redundancies:

```
insert into Students(StudentID, Name, Course, Grade,
                                             Program, Faculty)
values (1241, ’Mike’, ’Database Systems’, 1.7, ’EE’,  ’Mech.Eng.’)
```

  - Database Systems is assigned to the EE program:
    violates FD Course $\rightarrow$ Program
  - the EE program is assigned to the Mech.Eng. faculty:
    violates FD Program $\rightarrow$ Faculty

- **update** and **delete** anomalies exist, too.

Functional Dependencies

- functional dependency between sets of attributes X and Y of a given relation:

> if in each tuple of the relation the values in X determines the values in Y.

  - if two tuples don't differ regarding the attributes X, then they have also the same values for attributes Y
  - $t_1(X) = t_2(X) \Rightarrow t_1(Y) = t_2(Y)$
- Notation for functional dependencies (FD): $X \longrightarrow Y$

Primary Keys as Special Case of FDs

- Consider the example at slide 16
  StudentID $\rightarrow$ Name, Course, Grade, Program, Faculty
- It always holds: StudentID $\rightarrow$ StudentID, the entire schema at the right hand side
- if left hand side is minimal, then it is a key
- Formally: X is a key, if for relation schema R the FD $X \rightarrow R$ is satisfied and X is minimal

Goal of the Database Design: transform all given functional dependencies into „key dependencies“ without losing semantic information.

Deriving FDs

A	B	C
1	10	22
2	10	22
3	11	22
4	10	22

- satisfies $A \rightarrow B$ and $B \rightarrow C$
- thus, also $A \rightarrow C$ holds
- but, $C \rightarrow A$ and $C \rightarrow B$ do not hold
- $\{A \rightarrow B,\ B \rightarrow C\}$ implies $A \rightarrow C$
- $F = \{A \rightarrow B,\ B \rightarrow C\} \models A \rightarrow C$

Deriving FDs (cont.)

- Constructing the closure: Determine all functional dependencies, which can be derived from a given set of FDs
- Closure $F_R^+ := \{ f \mid (f \text{ FD on } R) \wedge F \vDash f \}$
- Example:
  $\{ A \longrightarrow B, B \longrightarrow C \}^+ = \{ A \longrightarrow B, B \longrightarrow C, A \longrightarrow C, AB \longrightarrow C, A \longrightarrow BC, \ldots, AB \longrightarrow AB, \ldots \}$

Inference Rules for FDs

F1	Reflexivity	$X \supseteq Y \Rightarrow X \rightarrow Y$
F2	Augmentation	$\{X \rightarrow Y\} \Rightarrow XZ \rightarrow YZ$ and $XZ \rightarrow Y$
F3	Transitivity	$\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow X \rightarrow Z$
F4	Decomposition	$\{X \rightarrow YZ\} \Rightarrow X \rightarrow Y$
F5	Union	$\{X \rightarrow Y, X \rightarrow Z\} \Rightarrow X \rightarrow YZ$
F6	Pseudo Transitivity	$\{X \rightarrow Y, WY \rightarrow Z\} \Rightarrow WX \rightarrow Z$

- F1-F3 also known as Armstrong-Axioms (sound, complete)
  - sound: generate only functional dependencies in the closure of a set of functional dependencies F^+
  - complete: repeated application of the rules will generate all functional dependencies in F^+
  - independent: none of the rules may be dropped

Example / Task

Given is the relational schema $\boldsymbol{R}(A, B, C, D, E)$

as well as the functional dependencies

$$AC \rightarrow BDE, B \rightarrow D, A \rightarrow E.$$

What is the key?

Schema Properties

- Choose relation schemas, keys and foreign keys in a way, such that
  1. all application data can be derived from the base relations,
  2. only semantically meaningful and consistent data can be represented,
  3. data is represented without redundancies.

- Now: requirement #3
  - redundancies in a single relation: normal forms
  - global redundancies: minimality

Normal Forms

- define properties of relation schemas
- forbid certain combinations of functional dependencies in a given relation
- avoid redundancies and update anomalies

1st Normal Form

- allows only atomic attributes in a relation schema, i.e., attribute values are from domains of basic data types like *integer* or *string*, but not constructors such as *array* or *set*

- The following relation violates 1NF:

StudentID	Name	Email
1234	Bob Stadler	bob@gmail.com, bob_s@yahoo.com
1235	Steve Palmer	steve@web.de
…	…	…

1st Normal Form (cont.)

StudentID	Name	Email
1234	Bob Stadler	bob@gmail.com
1234	Bob Stadler	bob_s@yahoo.com
1235	Steve Palmer	steve@web.de
…	…	…

2nd Normal Form

- partial dependency exists, if an attribute depends already on a subset of the key

StudentID	Name	Course	Grade	Program	Faculty
1234	Bob	Operating Systems	1.7	CS	Com. Science
1234	Bob	Database Systems	2.3	CS	Com. Science
1236	Mary	Analysis	1.0	EE	Electr. Eng.
…	…	…	…	…	…

  - f_1: StudentID, Course $\rightarrow$ Grade
  - f_2: StudentID $\rightarrow$ Name
  - f_3: Course $\rightarrow$ Program, Faculty
  - f_4: Program $\rightarrow$ Faculty

- 2NF removes such partially dependencies for non-key attributes

2nd Normal Form

<u>StudentID</u>	Name	<u>Course</u>	Grade	Program	Faculty
1234	Bob	Operating Systems	1.7	CS	Com. Science
1234	Bob	Database Systems	2.3	CS	Com. Science
1236	Mary	Analysis	1.0	EE	Electr. Eng.
...	...	...	...	...	...

- f_1: StudentID, Course $\rightarrow$ Grade
- f_2: StudentID $\rightarrow$ Name
- f_3: Course $\rightarrow$ Program, Faculty
- f_4: Program $\rightarrow$ Faculty

Decomposition (see also slide 52)

- Relation $r(R)$, K – key, attributes $A, B \subset R$,
- $A \rightarrow B$ **violates** normal form

Decomposition

1. new relation schema for attributes A, B from „disturbing“ dependency
2. determining attribute(s) A is the key for the new schema
3. determining attribute(s) A also stays in the old schema (as foreign key, and as a base for joining the relations)

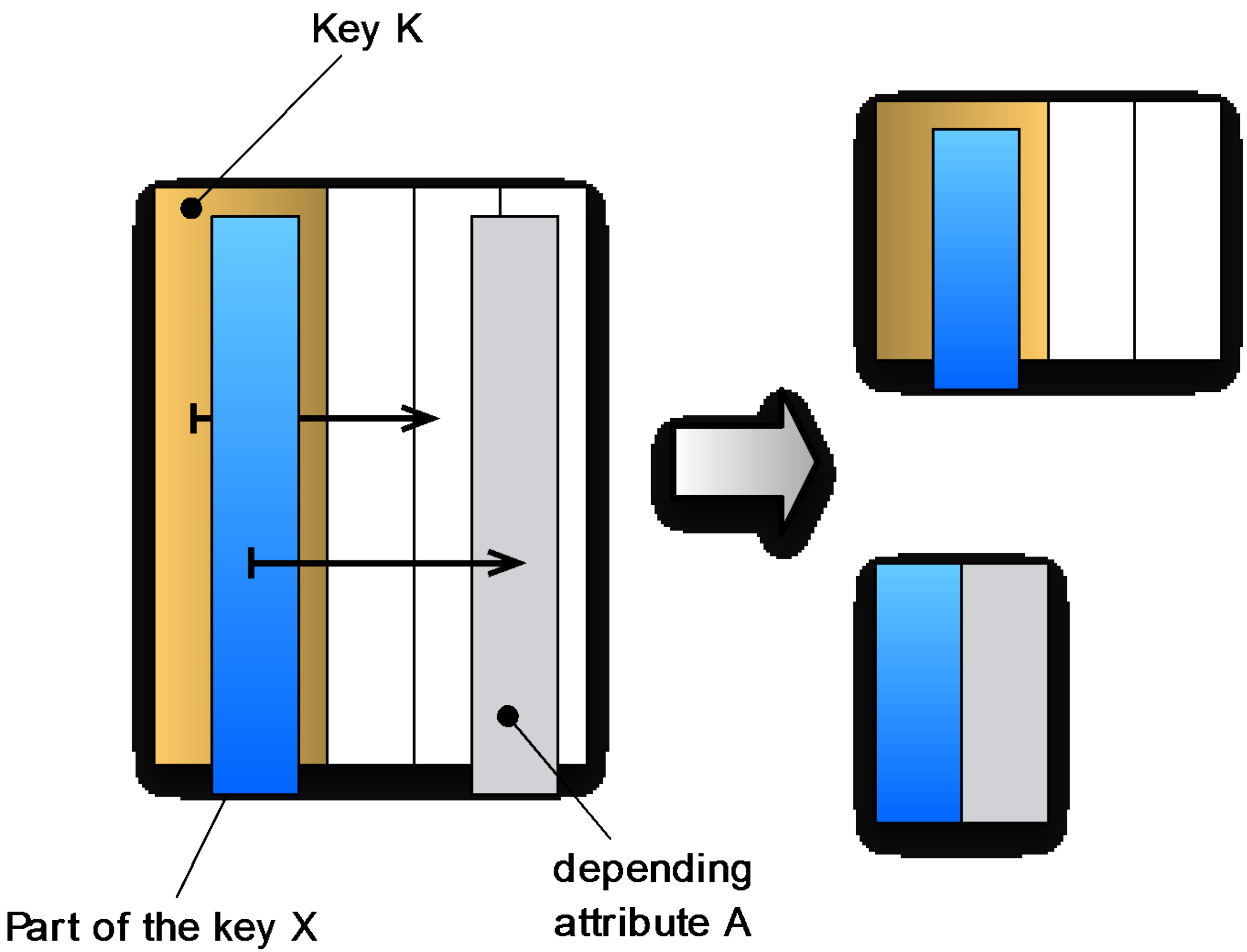
$\Rightarrow$ Result: Relations with schema

$R1 = R - B$ key: K

$R2 = A\ B$ key: A

$$\mathbf{r} = \pi_{R1}(\mathbf{r}) \bowtie \pi_{R2}(\mathbf{r})$$

Eliminating Partial Dependencies



2nd Normal Form (cont.)

Example relation in 2NF

- Student(<u>StudentID</u>, Name)
- Grades(<u>StudentID, Course</u>, Grade)
- Courses(<u>Course</u>, Program, Faculty)

<u>StudentID</u>	Name
1234	Bob
1236	Mary

<u>Course</u>	Program	Faculty
Operating Systems	CS	Com. Science
Database Systems	CS	Com. Science
Analysis	EE	Electr. Eng.

<u>StudentID</u>	<u>Course</u>	Grade
1234	Operating Systems	1.7
1234	Database Systems	2.3
1236	Analysis	1.0

2nd Normal Form (cont.)

- Note:
  - Partial depending attributes are only considered, if there are not prime attributes.
- Formal definition of 2NF:
  - extended relation schema $R = (R, \mathbf{K})$, a set of FD F over R

> R is in 2NF, if R is in 1NF and each non-prime attribute in R depends completely on the whole of every candidate key in R

- Note:
  - When a 1NF table has no composite key, it is automatically in 2NF.

3rd Normal Form

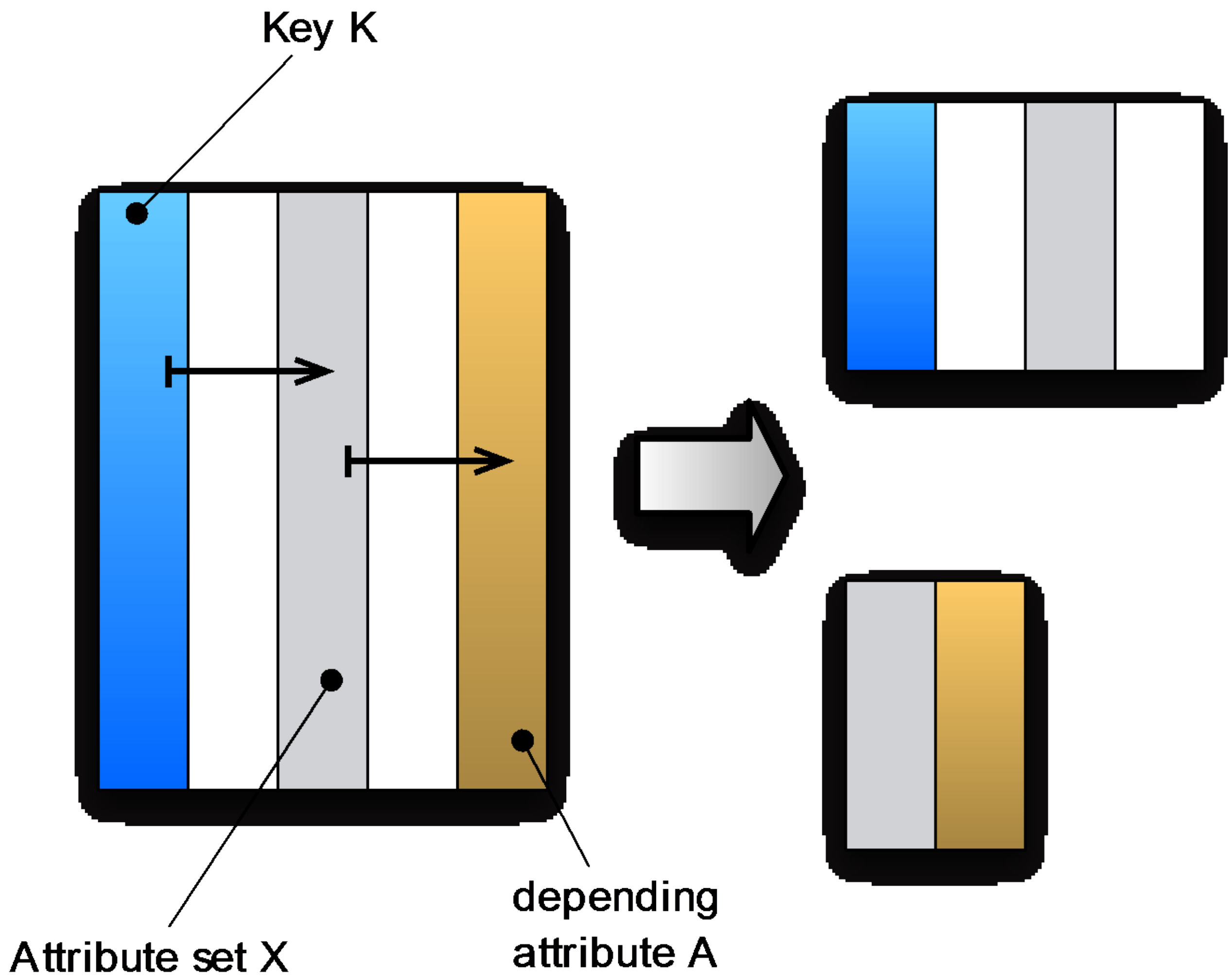
- Additionally removes transitive dependencies
  - e.g., Course→ Program and Program → Faculty in the example relation from slide 26
  - transitive dependency in Courses, i.e. Courses **violates** 3NF

Courses

Course	Program	Faculty
Operating Systems	CS	Com. Science
Database Systems	CS	Com. Science
Analysis	EE	Electr. Eng.

- Note:
  - for 3NF only non-key attributes on the right-hand side of transitive dependencies are considered

Eliminating Transitive Dependencies



3nd Normal Form (cont.)

Example relation in 3NF

- Student(<u>StudentID</u>, Name)
- Grades(<u>StudentID, Course</u>, Grade)
- Courses(<u>Course</u>, Program)
- Programs(<u>Program</u>, Faculty)

<u>Course</u>	Program
Operating Systems	CS
Database Systems	CS
Analysis	EE

<u>StudentID</u>	Name
1234	Bob
1236	Mary

<u>Program</u>	Faculty
CS	Com. Science
EE	Electr. Eng.

<u>StudentID</u>	<u>Course</u>	Grade
1234	Operating Systems	1.7
1234	Database Systems	2.3
1236	Analysis	1.0

3rd Normal Form: Formal Definition

- Formal definition of 3NF:
  - Relation schema R, $X \subseteq R$ and F is a set of FDs over R

- $A \in R$ transitively depends on X regarding F iff there exists $Y \subseteq R$ with $X \rightarrow Y, Y \nrightarrow X, Y \rightarrow A$, $A \notin XY$
- extended relation schema $R = (R, \boldsymbol{K})$ is in 3NF regarding F iff $\nexists A \in R$
  - A is a non-prime attribute in R
  - A is transitively dependent on $K \in \boldsymbol{K}$ wrt. F.

Exercise / Task (1)

- Given is the relational schema $\boldsymbol{R}(A, B, C, D, E)$

  as well as the functional dependencies

  $$AC \rightarrow BDE, B \rightarrow D, A \rightarrow E.$$

Design, step-by-step, a database schema in 3rd Normal Form using decomposition. Note the primary keys.

Exercise / Task (2)

Given is a booking relation in a flight booking system (assume attribute values are atomic):

BOOKING (FlightNr, CNr, Airline, Land, Destination, DestinationCountry, Departure, FlightDate, BookingDate, Price, CName, BonusMiles)

The following functional dependencies hold in the relation schema, leading to data redundancies:

FlightNr $\rightarrow$ Destination

FlightNr $\rightarrow$ Airline

FlightNr $\rightarrow$ DestinationCountry

FlightNr $\rightarrow$ Land

FlightNr $\rightarrow$ Departure

Destination $\rightarrow$ DestinationCountry

Airline $\rightarrow$ Land

CNr $\rightarrow$ CName

CNr $\rightarrow$ BonusMiles

Design, step-by-step, a database schema in 3rd Normal Form using decomposition. Note the primary keys.

Inference Rules for FDs

F1	Reflexivity	$X \supseteq Y$	$\Rightarrow X \to Y$
F2	Augmentation	$\{ X \to Y \}$	$\Rightarrow XZ \to YZ$ and $XZ \to Y$
F3	Transitivity	$\{ X \to Y,\ Y \to Z \}$	$\Rightarrow X \to Z$
F4	Decomposition	$\{ X \to YZ \}$	$\Rightarrow X \to Y$
F5	Union	$\{ X \to Y,\ X \to Z \}$	$\Rightarrow X \to YZ$
F6	Pseudo Transitivity	$\{ X \to Y,\ WY \to Z \}$	$\Rightarrow WX \to Z$

R	Reflexivity	$\{ \}$	$\Rightarrow X \to X$
A	Accumulation	$\{ X \to YZ, Z \to AW \}$	$\Rightarrow X \to YZA$
P	Transitivity	$\{ X \to YZ \}$	$\Rightarrow X \to Y$

Exercise / Task (3)

Given is a relation for a superstore (assume attribute values are atomic):

R (*ArtNo, SNo, Supplier, Phone, Article, Price, Unit, Store, Amount*)

The following functional dependencies hold in the relation schema, leading to data redundancies:

(1) $ArtNo, SNo \rightarrow Price, Amount$

(2) $ArtNo \rightarrow Article, Unit$

(3) $SNo \rightarrow Supplier, Phone$

(4) $Article \rightarrow Store$

Design, step-by-step, a database schema in 3rd Normal Form using decomposition. Note the primary keys.

Boyce-Codd Normal Form

- Stronger version of 3NF: Forbids transitive dependencies also between prime attributes

Product	Supplier	Retailer	Price
SkyPhone	Macrosoft	phone.com	600
Super Tablet	Ontel	superstore.de	400
SkyPhone	Pear	amazon.de	580
Surface Phone	Macrosoft	phone.com	400

  - FDs: Product, Supplier → Price,
    Supplier → Retailer, Retailer → Supplier
  - Candidate keys: { Product, Supplier }, { Product, Retailer }
- in 3NF, but not in BCNF

Boyce-Codd Normal Form (cont.)

- BCNF formally:
  - extended relation schema $R = (R, \boldsymbol{K})$, FD set F

$\nexists A \in R : A$ depends transitively on $K \in \boldsymbol{K}$ regarding F.

- BCNF can violate Dependency Preserving Decomposition, thus, we often stop at 3NF

Boyce-Codd Normal Form (cont.)

- Schema in BCNF:
  - PRODUCTS (Product, Supplier, Price)
  - RADERS (Supplier, Retailer)

Product	Supplier	Price
SkyPhone	Macrosoft	600
Super Tablet	Ontel	400
SkyPhone	Pear	580
Surface Phone	Macrosoft	400

Supplier	Retailer
Macrosoft	phone.com
Ontel	superstore.de
Pear	amazon.de

Minimality

- Goal:
  - Avoid global redundancies
  - Achieve other criteria (such as normal forms) with the minimal number of schemas

- Example:
  - attribute set ABC, FDs $\{A \to B, B \to C\}$
  - database schema in 3NF:

$$S = \{(AB, \{A\}), (BC, \{B\})\}$$

$$S‘ = \{(AB,\{A\}),(BC,\{B\}),(AC,\{A\})\}$$

- but redundancies in $S‘$

Transformation Properties

- A decomposition of a relation into multiple relations should
  1. allow only consistent and semantically meaningful data (Dependency Preserving Decomposition) and
  2. allow to reconstruct the original data from the base relations (Lossless Decomposition)

Dependency Preserving Decomposition

- A set of functional dependencies can be transformed into an equivalent set of other dependencies
- specifically: into a set of key dependencies which can be easily checked by the DBMS
  - the set of dependencies has to be equivalent to the set of key dependencies in the resulting database schema
  - equivalence ensures, that the key dependencies semantically express the same integrity constraints as the original functional dependencies

Dependency Preserving Decomposition: Example

- Decomposing relation schema (slide 29) into 3NF:
  - Student(<u>StudentID</u>, Name)
  - Grades(<u>StudentID, Course</u>, Grade)
  - Courses(<u>Course</u>, Program)
  - Programs(<u>Program</u>, Faculty)
- with key dependencies
  - StudentID $\to$ Name
  - StudentID, Course $\to$ Grade
  - Course $\to$ Program
  - Program $\to$ Faculty
- is equivalent to FDs $f_1 \ldots f_4$ (slide 29)

$\leadsto$ dependency preserving decomposition

Dependency Preserving Decomposition: Example (2)

- Address data

  Zipcode (Z), City (C), Street (S), Number (N)
- and functional dependencies F

  $CSN \rightarrow Z,\ Z \rightarrow C$
- For a database schema S consisting of the single relation schema $(CSNZ,\ \{CSN\})$,
- is the set of key dependencies

  $\{\ CSN \rightarrow CSNZ\ \}$
- not equivalent to F and therefore S not dependency preserving

Dependency Preserving Decomposition (formal)

- Formal definition:
  - locally extended database schema $S = \{(R_1, K_1), \ldots, (R_p, K_p)\}$; set F of local dependencies

S characterizes completely F (or: is dependency preserving regarding F) iff

$$F \equiv \{ K \rightarrow R \mid (R, \mathbf{K}) \in S, K \in \mathbf{K} \}$$

Lossless Decomposition

- To achieve the criteria of normal forms, the relation schemas have to be decomposed into smaller relation schemas.
- Allow only "meaningful" decompositions which allow to reconstruct the original relation from the decomposed relations using a natural join $\rightsquigarrow$ Lossless Decomposition

Losless Decomposition: Examples

- Decomposition of relation schema $R = ABC$ in

$$R_1 = AB \text{ and } R_2 = BC$$

- Decomposition is not lossless for functional dependencies

$$F = \{A \rightarrow B,\ C \rightarrow B\}$$

- But is lossless in case of

$$F' = \{A \rightarrow B,\ B \rightarrow C\}$$

Losless Decomposition: Examples

- Original relation:

A	B	C
1	2	3
4	2	3

- Decomposition:

A	B
1	2
4	2

B	C
2	3

- Join:

A	B	C
1	2	3
4	2	3

Non-Lossless Decomposition

- Original relation:

A	B	C
1	2	3
4	2	5

- Decomposition:

A	B
1	2
4	2

B	C
2	3
2	5

- Join:

A	B	C
1	2	3
4	2	3
1	2	5
4	2	3

Lossless Decomposition (formal definition)

- Formal definition:

A decomposition of an attribute set X in $X_1,\ldots,X_p$ with $X = \bigcup_{i=1}^{p} X_i$ is lossless regarding a set of functional dependencies F over X iff

$$\forall r \in \mathbf{SAT}_X(F) : \pi_{X_1}(r) \bowtie \cdots \bowtie \pi_{X_p}(r) = r$$

- basic criteria for lossless decomposition of two relation schema:

  Decomposition of X in X_1 and X_2 is lossless wrt. F,
  if $X_1 \cap X_2 \rightarrow X_1 \in F^+$ or $X_1 \cap X_2 \rightarrow X_2 \in F^+$

Database Design Algorithms: Goals

- Given: universe U and set of FD F
- derive locally extended database schema $S = \{(R_1,K_1),\ldots,(R_p,K_p)\}$ with
  - Dependency Preserving Decompostion: S characterizes F completely
  - 3rd Normal Form: S is in 3NF wrt. F
  - Lossless Decomposition: Decomposition of U in $R_1,\ldots, R_p$ is lossless wrt. F
  - Minimality: $\nexists S' : S'$ satisfies all these properties and $|S'| < |S|$

Database Design Algorithms: Summary

- 3rd Normal Form
  - to avoid local redundancies
- Minimality
  - to avoid global redundancies
- Dependency Preserving Decompostion
  - allow only consistent and semantically meaningful data
  - A set of functional dependencies can be transformed into an equivalent set of other dependencies (key dependencies)
- Lossless Decomposition:
  - Allow only "meaningful" decompositions which allow to reconstruct the original relation from the decomposed relations using a natural join

Decomposition

- Given: initial universal relation schema $R = (U, K(F))$ consisting of all attributes, a set of FDs F over R, and a set of keys implied by F
  - set of attributes U and a set of FD F
  - find all $K \rightarrow U$ where K is minimal and where for $K \rightarrow U \in F^+$ holds $(K(F))$
- Look for a decomposition in $D = \{R_1, R_2, \ldots\}$ where all R_i in 3NF

Decomposition: Algorithm

DECOMPOSE(R)

Let $D := \{R\}$

while $R' \in D$, which violates 3NF

/ Find attribute A, which depends on K transitively */*

if Key K with $K \rightarrow Y, Y \nrightarrow K, Y \rightarrow A, A \notin KY$

then

/ Decompose relation schema R regarding A */*

$R_1 := R - A,\ R_2 := YA$

$R_1 := (R_1, K),\ R_2 := (R_2, K_2 = \{Y\})$

$D := (D - R') \cup \{R1\} \cup \{R2\}$

end if

end while

return D

Decomposition: Example

- Initial relation schema $R = ABC$
- Functional dependencies $F = \{A \rightarrow B, B \rightarrow C\}$
- Key $K = A$

$$R_1 = A\,B \;,\; K_1 = A$$
$$R_2 = B\,C \;,\; K_2 = B$$

Query Languages

- A query language (QL) is a language that allows users to manipulate and retrieve data from a database.

- The relational model supports simple, powerful QLs (having strong formal foundation based on logics, allow for much optimization)

- Query Language != Programming Language
  - QLs are not expected to be turing complete, not intended to be used for complex applications/computations
  - QLs support easy access to large data sets
- Categories of QLs: procedural versus declarative

Query Languages (cont.)

- Two (mathematical) query languages form the basis for "real" languages (e.g., SQL) and for implementation
  - Relational Algebra: procedural, very useful for representing query execution plans, and query optimization techniques.
  - Relational Calculus: declarative, logic based language

- Understanding algebra (and calculus) is the key to understanding SQL, query processing and optimization.

Relational Algebra

- Procedural language
- Queries in relational algebra are applied to relation instances, result of a query is **again a relation** instance
- Six basic operators in relational algebra:
  - select: selects a subset of tuples from relation
  - project: deletes unwanted columns from relation
  - join: allows to combine two relations
  - union: tuples in relation 1 plus tuples in relation 2
  - set difference: tuples in relation 1, but not in relation 2
  - rename: renames attribute(s) and relation
- The operators take one or two relations as input and give a new relation as a result (relational algebra is "closed").

Select Operator

- Notation: $\sigma_P(r)$
- Defined as

  $$\sigma_P(r) := \{\, t \mid t \in r \text{ and } P(t) \,\}$$

- where
  - r is a relation (name),
  - P is a formula in propositional calculus, composed of conditions of the form

    <*attribute*> = <*attribute*> or <*constant*>
  - Instead of "=" any other comparison predicate is allowed ($\neq$, $<$, $>$ etc).
  - Conditions can be composed through $\wedge$ (and), $\vee$ (or), $\neg$ (not)

Select Operator: Example

- Given the relation r

A	B	C
1	1	7
2	2	3
3	3	6
4	5	9

- Query: $\sigma_{A=B \wedge C>5}(r)$

A	B	C
1	1	7
3	3	6

Project Operator

- Notation: $\pi_{A1,A2,\ldots,Ak}(r)$
  where $A_1, \ldots, A_k$ are attributes and r is a relation (name)
- The result of the projection operation is defined as the relation that has k columns obtained by erasing all columns from r that are not listed.
- Duplicate rows are removed from result because relations are sets.

Project Operator: Example

- Example: given the relation r

A	B	C
1	2	3
1	3	3
2	3	4

- Query: $\pi_{A,C}\,(r)$

A	C
1	3
2	4

Join Operator

- Notation (natural join): $r_1 \bowtie r_2$

  where r_1 and r_2 are relation (names)
- combines two relations on same values in attributes with the same names

  $$r_1 \bowtie r_2 \quad := \quad \{t \mid t(R_1 \cup R_2) \wedge [\forall i \in \{1,2\} \exists t_i \in r_i : t_i = t(R_i)]\}$$

- resulting schema for $r_1 \bowtie r_2$ with r_1 has schema R_1 and r_2 has schema R_2 is union of attribute sets $R_1 \cup R_2$
- for $R_1 \cap R_2 = \{\}$ it holds: $r_1 \bowtie r_2 = r_1 \times r_2$

Join Operator: Example

- Example: given the relations r and s

A	B
11	1
12	2
14	4

B	C
1	21
2	22
3	23

- Query: $r \bowtie s$

A	B	C
11	1	21
12	2	22

Union Operator

- Notation: $r \cup s$, where both r and s are relations
- Defined as $r \cup s := \{ t \mid t \in r \text{ or } t \in s \}$
- For $r \cup s$ to be applicable,
  - r, s must have the same number of attributes
  - attribute domains must be compatible (e.g., 3rd column of r has a data type matching the data type of the 3rd column of s)

Union Operator: Example

- Example: given the relations r and s

r

A	B
11	1
11	2
12	1

s

A	B
11	2
12	3

$r \cup s$

A	B
11	1
11	2
12	1
12	3

Set Difference Operator

- Notation: r – s where both *r* and *s* are relations
- Defined as $r - s := \{ t \mid t \in r \text{ and } t \notin s \}$
- For $r - s$ to be applicable,
  - r and s must have the same arity
  - attribute domains must be compatible

Set Difference Operator: Example

- Example: given the relations *r* and *s*

r

A	B
11	1
11	2
12	1

s

A	B
11	2
12	3

r - *s*

A	B
11	1
12	1

Rename Operator

- Allows to name and therefore to refer to the result of relational algebra expression.
- Allows to refer to a relation by more than one name (e.g., if the same relation is used twice in a relational algebra expression).
  - Example:
    $\rho_x(E)$
    returns the relational algebra expression E under the name x
- If a relational algebra expression E (which is a relation) has the arity k, then
  $\rho_{x(A,B,\ldots)}(E)$
  returns the expression E under the name x, and with the attribute names $A, B, \ldots$.

Composition of Operators

- It is possible to build relational algebra expressions using multiple operators

  similar to the use of arithmetic operators (nesting of operators)
- A **basic expression** in the relational algebra consists of either of the following:
  - a relation in the database
  - a constant relation (fixed set of tuples, e.g., {(1, 2), (1, 3), (2, 3)})
- If E_1 and E_2 are expressions of the relational algebra, then the following expressions are relational algebra expressions, too:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \bowtie E_2$
  - $\sigma_P(E_1)$ where P is a predicate on attributes in E_1
  - $\pi_A(E_1)$ where A is a list of some of the attributes in E_1
  - $\rho_x(E_1)$ where x is the new name for the result relation [and its attributes] determined by E_1

Example Queries

- Assume the following relations:
  - BOOKS(DocId, Title, Publisher, Year)
  - STUDENTS(StId, StName, Major, Age)
  - AUTHORS(AName, Address)
  - borrows(DocId, StId, Date)
  - has-written(DocId, AName)
  - describes(DocId, Keyword)

Example Queries

- Q1: List the year and title of each book.
- Q2: List all information about students whose major is CS.
- Q3: List all students with the books they can borrow.
- Q4: List all books published by Springer after 2010.
- Q5: List the name of those authors who are living in Kazan.
- Q6: List the name of students who are older than 30 and who are not studying CS.
- Q7: Rename AName in the relation AUTHORS to Name.

Example Queries (cont.)

- Q8: List the names of all students who have borrowed a book and who are CS majors.
- Q9: List the title of books written by the author „Silberschatz“.
- Q10: As Q9, but not books that have the keyword „database“.
- Q11: Find the name of the youngest student.
- Q12: Find the title of the oldest book.

Additional Operators

- These operators do not add any power (expressiveness) to the relational algebra but simplify common (often complex and lengthy) queries.

- Set Intersection $\cap$
- Condition Join / Theta Join $\bowtie_c$
- Division $\div$

Set Intersection

- Notation: $r \cap s$
- Defined as $r \cap s := \{ t \mid t \in r \text{ and } t \in s \}$
- For $r \cap s$ to be applicable,
  - r and s must have the same arity
  - attribute domains must be compatible
- Derivation: $r \cap s = r - (r - s)$

- Example: given the relations r and s

r

A	B
11	1
11	2
12	1

s

A	B
11	2
12	3

$r \cap s$

A	B
11	2

Theta Join

- Notation: $r \bowtie_C s$
- C is a condition on attributes in $R \cup S$
- result schema is the same as that of Natural Join
  - If $R \cap S \neq \emptyset$; and condition C refers to these attributes, some of these attributes must be renamed.
  - C is sometimes also called θ.
- Derivation: $r \bowtie_C s = \sigma_C (r \times s)$
  - Note that C is a condition on attributes from both r and s
  - If C involves only the comparison operator „=“ the theta join is also called *Equi-Join.*

Theta Join: Example

- given the relations r and s

r

A	B	C
1	2	3
4	5	6
7	8	9

s

D	E
3	1
6	2

$r \bowtie_{B<D} s$

A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

Division

- Notation: $r \div s$
  - Precondition: attributes in S must be a subset of attributes in R, i.e., $S \subseteq R$.
- Let r, s be relations on schemas R and S, respectively, where
  - $R(A_1,\ldots,A_m,B_1,\ldots,B_n)$
  - $S(B_1,\ldots,B_n)$
- The result of $r \div s$ is a relation on schema
  $R - S = (A_1, \ldots, A_m)$
- The result of the division operator consists of the set of tuples from r defined over the attributes $R - S$ that match the combination of every tuple in s.
- $r \div s := \{ t \mid t \in \pi_{R-S}(r) \wedge \forall u \in s: tu \in r \}$
- Suited for queries that include the phrase "for all".

Division: Example

- given the relations r and s

r

A	B	C	D	E
11	a	2	a	1
11	a	4	a	1
11	a	4	b	1
12	a	4	a	1
12	a	4	b	3
13	a	4	a	1
13	a	4	b	1
13	a	3	b	1

s

D	E
a	1
b	1

$r \div s$

A	B	C
11	a	4
13	a	4

More Queries

- Q13: List each book with its keywords.
- Q14: List each student with the books s/he has borrowed.
- Q15: List the title of books written by the author „Ullman“.
- Q16: List the authors of the books the student „Peter“ has borrowed.
- Q17: Which books have both keywords „database“ and „programming“?

Summary

- The relational model provides a solid theoretical foundation for RDBMS.
- The goal of database design is to eliminate redundancies by normal form criteria.
- Normalization requires the identification and derivation of functional dependencies.
- Relational algebra is a procedural (mathematical) language for formulating queries on relations.

Exercise

Customers

CNo	Name
123	Müller, F
456	Abel, M
789	Schulz, R
109	Jahn, E

Products

PNo	Description
45	Butter
56	Wine
11	Milk
67	Oranges
13	Potatoes

Stores

SID	Name	Address
27	Aldi	Huettenholz
23	Netto	Herderstrasse
24	Tegut	Goethepassage
20	Rewe	Muehlgraben

Receipts

RNo	PNo	Quantity
1	45	2
1	67	10
3	11	2
5	67	5
7	56	1
7	67	11
9	45	1
9	56	3
9	67	7

Special_Offers

SID	PNo
27	13
27	56
23	67
23	13
24	56
27	67
24	67

Sales

RNo	SID	Date	Time	CNo
1	23	27.09.	08:13	456
3	20	30.09.	09:59	123
5	24	18.10.	12:07	789
7	27	19.10.	10:43	456
9	27	19.10.	21:01	123
17	20	06.12.	11:34	403

Task (1)

Give a verbal formulation of the following algebra expressions. What is the result of each expression?

a) π_{RNo} (*Receipts*)

b) π_{Name} (*Stores* $\bowtie$ *Sales*)

c) π_{SID} (*Stores*) – π_{SID} (*Special_Offers*)

d) π_{CNo} (($\sigma_{time < 10:00}$ (*Sales*)) $\cup$ ($\sigma_{time > 19:00}$ (*Sales*)))

e) *Customers* – $\pi_{CNo,\ Name}$ (*Customers* $\bowtie$ *Sales*)

f) π_{SID} (*Stores*)

– π_{SID} [(π_{SID} (*Stores*) $\times$ π_{ANo} (*Receipts*)) – *Special_Offers*]

Task (2)

Formulate the following queries in relational Algebra

- Provide a list of all customer names!
- Find all receipts of sales that took place in the morning!
- Which products were bought after 7PM?
- Give the names of customers together with their purchased products!
- Which customers have not bought anything?
- Which stores have the fewest products on sale?
- Which store had the earliest sale?

Given: $r_1(R_1)$, $r_2(R_2)$ with $R_2 \subseteq R_1$, $R' = R_2 - R_1$

$$r'(R') = \{ t \mid \forall\, t_2 \in r_2 \;\; \exists\, t_1 \in r_1 : \; t_1(R') = t \;\wedge\; t_1(R_2) = t_2 \}$$

$$= \; r_1 \div r_2$$

$$= \; \pi_{R'}(r_1) - \pi_{R'}(\, (\, \pi_{R'}(r_1) \;\bowtie\; r_2 \,) - r_1 \,)$$

Task (1) - Solution

Give a verbal formulation of the following algebra expressions. What is the result of each expression?

a) The numbers (RNo) from all receipts (without duplicates)

b) The name of the stores, where was purchased

c) The SID of the stores, which have no special offers

d) The customer numbers (CNo) of customers who have purchased early in the morning or late in the evening

e) The data (CNo, Name) of customers who have purchased nothing

f) Stores that have **all** purchased products on special offer