Introduction to Databases
(Winter Term 2002/2003)

(c) PD Dr. Wolfgang May
Universitäten Freiburg & Göttingen, Germany

`may@cs.uni-goettingen.de`

Wahlpflichtvorlesung Informatik; 2 SWS, 4 Kreditpunkte.
In dem Kurs werden nicht alle Abschnitte besprochen. Die prüfungsrelevanten Folien sind auf der Homepage der Vorlesung zu den einzelnen Terminen abgelegt. Die kompletten Folien zum Theorieteil und zum SQL-Teil sind auf der Homepage weiter unten zu finden.

Table of Contents

1 Basic Notions **1**

2 Data Models **10**

2.1 Entity Relationship Model (ERM) 12

2.2 Relational Model (RM) 43

3 Database Design **56**

3.1 Database Design: Design Steps 57

3.2 Logical Schema: Mapping ERM to RM 58

4 Relational Database Languages: Relational Algebra **65**

4.1 Relational Algebra: Computations over Relations 67

4.2 SQL 105

5 **Efficiency and Optimization** **113**
5.1 Efficient Data Access . . . 116
5.2 Efficient Query Evaluation . . . 127

6 **Running a Database: Safety and Correctness Issues** **156**
6.1 Transactions: Properties and Basic Notions . . . 157
6.2 Transaction models . . . 159
6.3 Multi-User Aspects . . . 161
6.4 Serializability . . . 166
6.5 Locks . . . 180
6.6 Safety: Error Recovery . . . 193

7 **Design Theory of the Relational Model** **203**
7.1 Functional Dependencies . . . 208
7.2 Decomposition of Relation Schemata . . . 223
7.3 Normal Forms based on FDs . . . 234
7.4 Join Dependencies and Multivalued Dependencies . . . 250

7.5 Summary . . . 256
7.6 Inclusion Dependencies . . . 257
7.7 Schema Design . . . 258

8 **Relational Database Languages: Relational Calculus** **260**
8.1 First-Order Logic and the Relational Calculus . . . 261
8.2 Formulas as Queries . . . 279
8.3 Comparison of the Algebra and the Calculus . . . 282
8.4 Equivalence of Algebra and (safe) Calculus . . . 290

Overview & Summary **300**

Chapter 1
Basic Notions

Context and Overview

- databases are used in ... economy, administration, research ...
- originally: storage of information
  relational model, SQL
- evolution: information systems, combining databases and applications
- today: Web-based information systems, electronic data exchange
  $\rightarrow$ new challenges, semistructured data, XML

Application Programs vs. Databases

(Application) Programs	**Databases**
Runtime Environment	Persistent Storage + Access

• short-lived computation	• long-lived model of an application domain
	  • schema
	  • data
	• temporary connections/access by application programs

APPLICATION PROGRAMS VS. DATABASES

(Application) Programs	Databases
Runtime Environment	Persistent Storage + Access
Programming Paradigms	
value-oriented	set-oriented, large amounts of data
variables	implicitly specified sets, iterators
procedural/imperative Pascal, C, C++, Java	declarative SQL
note: in both cases, object-orientation is added:	
Java: OO + imperative core	OQL: SQL + OO

APPLICATION PROGRAMS VS. DATABASES

(Application) Programs	Databases
Runtime Environment	Persistent Storage + Access
Operating Modes	
single-user	multiuser • user accounts
one-thread	concurrency

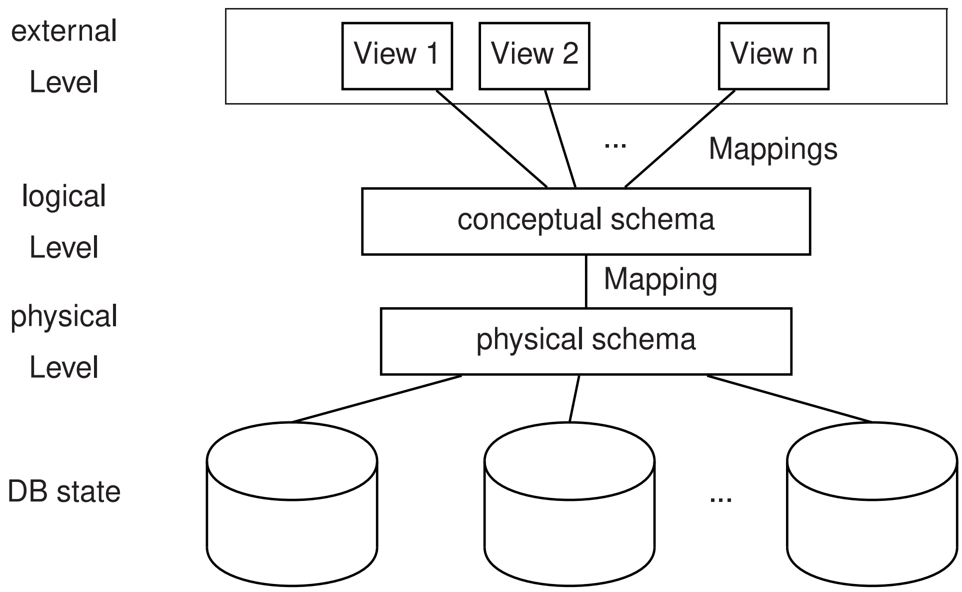
	• transactions
	• safety
	  • access control
	  • against physical failure
	• consistency, integrity

Application Programs vs. Databases

(Application) Programs	Databases
user-defined data structures	fixed data model user-defined schema internal storage aspects
small runtime data	large persistent data
program/algorithm	query
algorithms	internal algorithms
	transactions & safety

- A database system is a specialized data structure, with specialized behavior and -in contrast to most other data structures- specialized programming languages.
- (Note: the same holds for the XML data model.)

3-Level Architecture of a DBMS (ANSI/SPARC)



- global model of the application domain (conceptual schema)

Schema Levels

conceptual schema: The conceptual schema defines the model of the world as represented in the database, using an *abstract* formalism:

- definition of all relevant *object types* and *relationship types*,
- including *integrity constraints*,
- independent from the implementation.
- changes only rarely after being defined once.

Physical (internal) schema: Data structures for storing the data, and additional auxiliary structures for more efficient data handling (e.g., indexes).

Views/external schema/subschemata: Depending on the needs of special users, required object types and relationship types can be defined, derived from the ones that are defined in the *conceptual* model.

Mappings:

- define how the objects of the conceptual level are mapped to the physical level.
- define how the objects of the external level are defined based on those of the conceptual schema.

Schema and State

On each level, there exist the notions of *schema*, and *state*:

Database schema: the schema contains the *metadata* of the database, i.e., describes the concepts (e.g., object types and relationship types).

Database state: the state of a database (system) is given by the set of all data contained in the system. It represents the objects and relationships that hold in the application domain at a given timepoint.

With the time passing, a database passes through several database states.

- The admissible states are defined in terms of the conceptual schema (e.g., by integrity constraints),
- the database state itself is represented in the physical schema,
- users may access it through their views, using the external schema.

Data Dictionary: contains the definitions and mappings of the schemas.

Data Independence

Independence of the three levels:

- levels connected by mappings,
- every level may use a different data model,
- every level can be changed without affecting the others.

logical data independence: Changes and restructurings in the conceptual schema can be hidden against the external schema (by appropriate redefinition of mappings).

physical data independence: Modifications in the internal schema (splitting a table, adding an index, etc.) do not effect the conceptual schema (only redefinition of the mappings).

Chapter 2
Data Models

A *data model* defines modeling (specification-) constructs which can be used for modeling an application domain (in general, both its (static) data structures and its (dynamic) behavior).

- definition of *data structures* (object types and relationship types),
- definition of *integrity constraints*,
- definition of *operations* and their effects.

A data model consists of

- a **D**ata **D**efinition **L**anguage (DDL) for defining the schema: object types, relationship types, and integrity constraints.
- a **D**ata **M**anipulation **L**anguage (DML) for processing *database states* (inserting and modifying data)

  Operations are *generic operations* (querying, inserting, modifying, and deleting objects or relationships), or *procedures* that are constructed from basic operations.

Data Models

... there are different data models and modeling languages ...

- Entity-Relationship-Model (only static concepts)
- Unified Modeling Language (UML)
  comprehensive formalism for specifying processes, based on the object-oriented model
- Relational Model (very basic model)

2.1 Entity Relationship Model (ERM)

2.1.1 Main Structural Concepts

The main structural concepts for describing a schema in the ERM are **Entities** and **Relationships**.

Entity Types

Entity type: An entity type represents a concept in the real world. It is given as a pair $(E, \{A_1, \ldots, A_n\})$, where E is the name and $\{A_1, \ldots, A_n\},\ n \geq 0$ are the attributes (value properties) of a type.

Attribute: a relevant property of entities of a given type. Each attribute can have *values* from a given *domain*.

Example 2.1
(Continent, $\{$name, area$\}$)
(City, $\{$name, population, longitude, latitude$\}$),
(Province, $\{$name, area, population$\}$), □

Entities

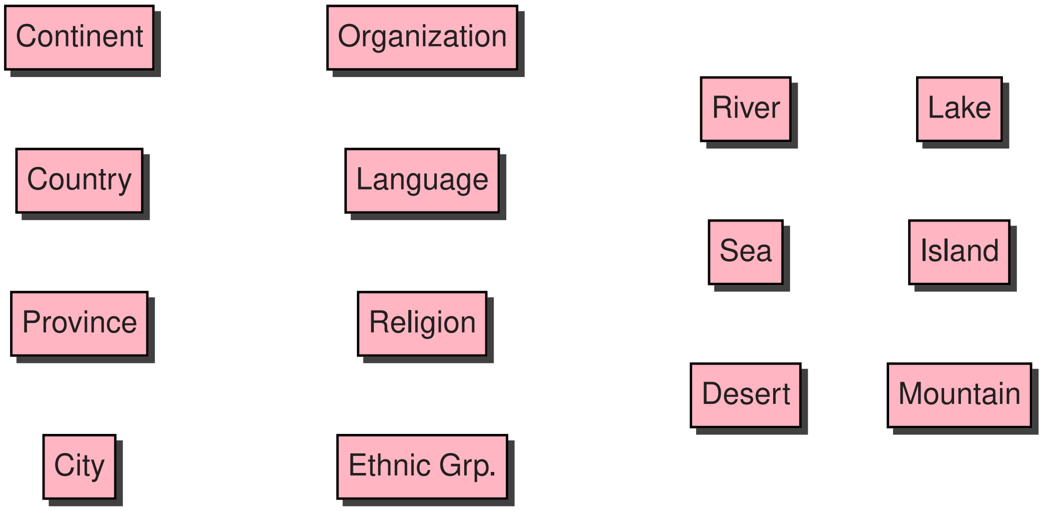
- An **entity set** e of an entity type E is a finite set of entities.
- each **entity** describes a real-world object. Thus, it must be of one of the defined entity types E. It assigns a value to each attribute that is declared for the entity type E.

Example 2.2
Entity set of the entity type (City, $\{$name, population, longitude, latitude$\}$):

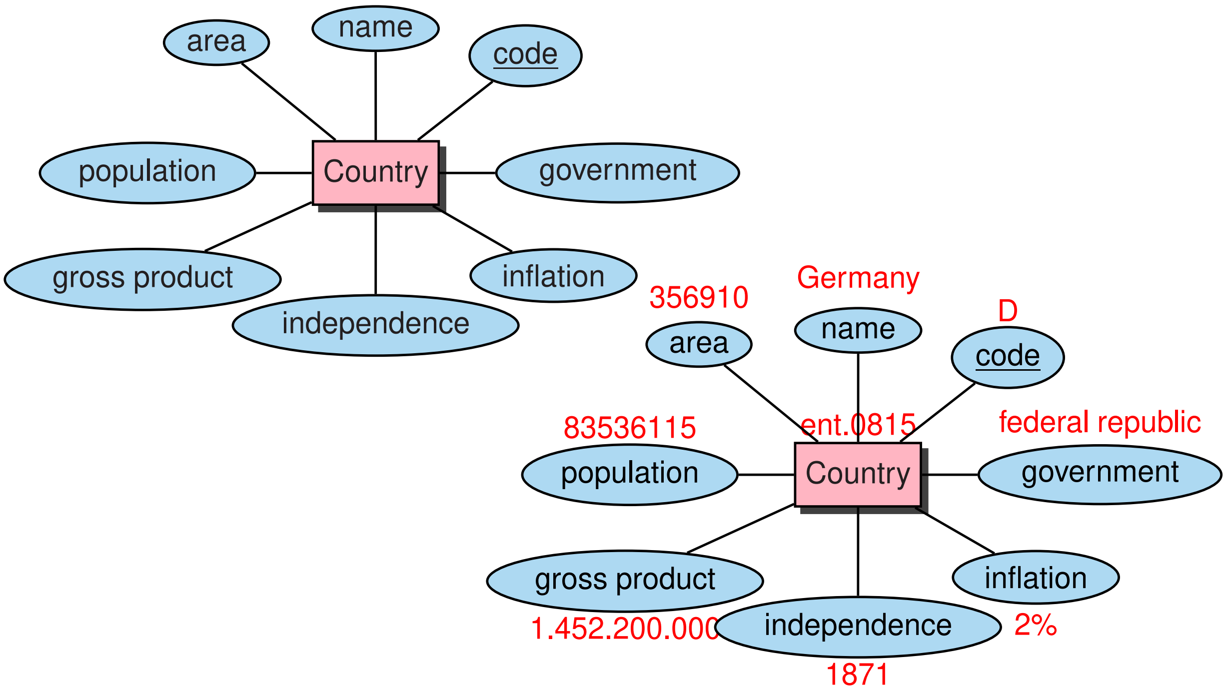
$\{$(name: Aden, population: 250000, longitude: 50, latitude: 13),
(name: Katmandu, population: 393494, longitude: 85,25, latitude: 27,45),
(name: Ulan_Bator, population: 479500, longitude: 107, latitude: 48)$\}$ □

Graphical Representation

- Entity types are represented as rectangles



- Attributes are represented as ovals



RELATIONSHIP TYPES

Relationship type: describes a concept of relationships between entities. It is given as a triple $(B, \{RO_1 : E_1, \ldots, RO_k : E_k\}, \{A_1, \ldots, A_n\})$, where B is the name, $\{RO_1, \ldots, RO_k\},\ k \geq 2$, is a list of *roles*, $\{E_1, \ldots, E_k\}$ is a list of entity types associated to the roles, and $\{A_1, \ldots, A_n\}, n \geq 0$ is the set of attributes of the relationship type.

In case that $k = 2$, the relationship type is called **binary**, otherwise n**-ary.**

Roles are pairwise different – the associated entity types are not necessarily pairwise distinct. In case that $E_i = E_j$ for $i \neq j$, there is a **recursive** relationship.

As long as there are no disambiguities, a role may be identified with the corresponding entity type. Roles are useful e.g. for annotating the semantic aspects of the reality.

Attributes describe relevant properties of relationships of a given type.

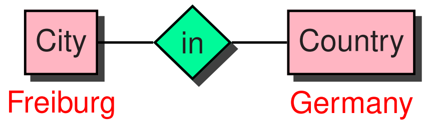
Example 2.3
(Capital, {City, Country}, ∅),
(encompasses, {Continent, Country}, {percent}),
(belongsto, {Province, Country}, ∅),
(flowsinto, {tributary: River, main: River}, ∅) □

RELATIONSHIP TYPES (CONT'D)

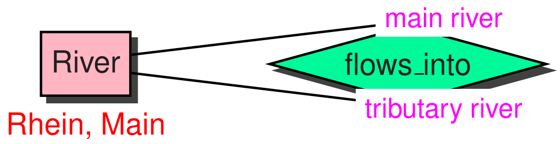
- A **relationship set** b of a relationship type B is a finite set of relationships.
- A **relationship** of a relationship type B is defined by the entities that are involved in the relationship, according to their associated roles. For each role, there is exactly one entity involved in the relationship, and every attribute is assigned a value.

(to be extended with examples)

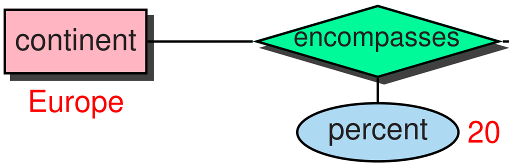
RELATIONSHIPS



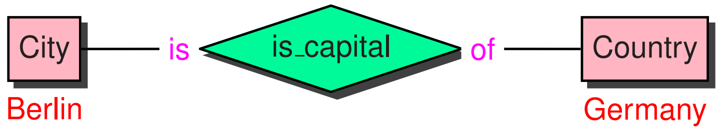
recursive relationship



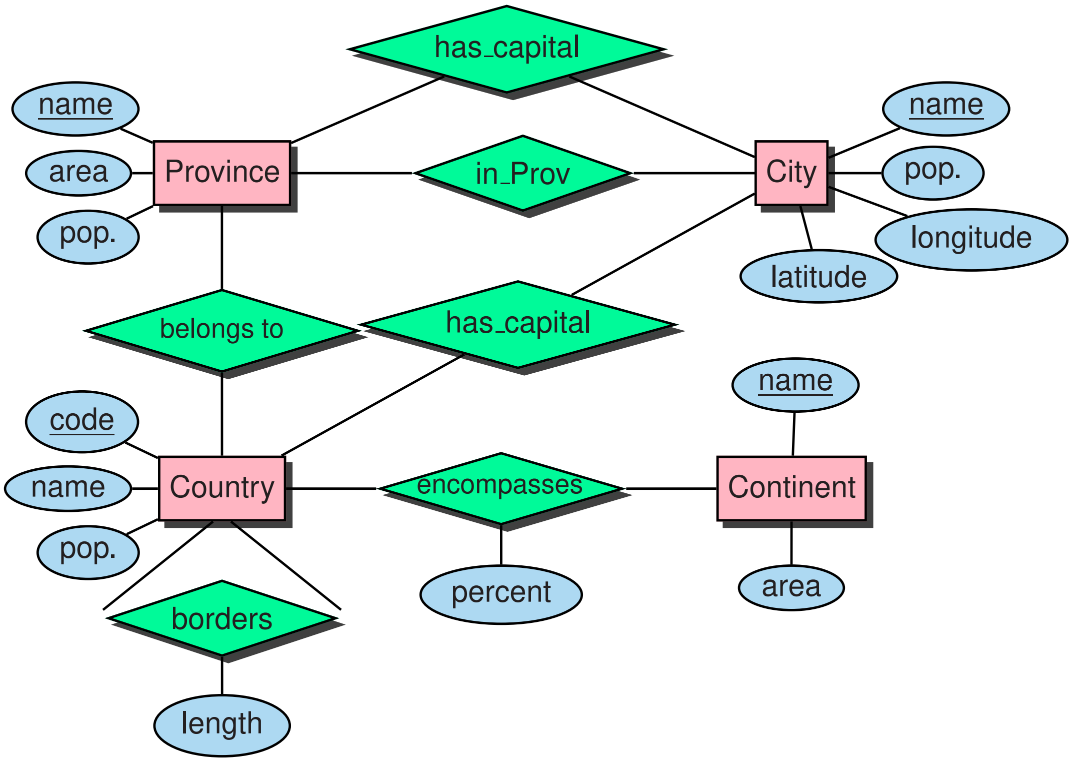
relationship with attributes



relationship with roles



Example: ER Model of a geographical database



DATABASE STATES

A **(database) state** associates the entity types and relationship types of a given schema with an **entity set** and a **relationship set**, respectively.

(cf. examples above – can be represented graphical as a graph/network)

2.1.2 Integrity Constraints

There are additional constraints on the admissible *database states*.

Domains: Every attribute is assigned a domain which specifies the set of admissible values.

Keys: a *key* is a set of attributes of an entity type, whose values together allow for a unique identification of all amongst all entities of a given type (cf. *candidate keys, primary keys*).

Relationship Complexities: every relationship type is assigned a complexity that specifies the minimal and maximal number of relationships in which an entity of a given type/role may be involved.

Referential Integrity: each entity which occurs in a relationship in any database state must also exist in the entity set of this state
(condition is trivial when represented as a graph, but crucial later in the relational model)

... to be described in detail on the following slides

Keys

A *key* is a set of attributes of an entity type, whose values together allow for a unique identification of all amongst all entities of a given type (cf. *candidate keys, primary keys*).

For an entity type $(E, \{A_1, \ldots, A_n\})$ and an entity set e of E, a set $K \subseteq \{A_1, \ldots, A_n\}$ satisfies the **key constraint** if:

- K uniquely **identifies** any element $\mu \in e$, i.e., for all $\mu_1, \mu_2 \in e$, if μ_1 and μ_2 have the same values for all attributes in K, then $\mu_1 = \mu_2$.

Declaring a set of attributes to be a key thus states a condition on all admissible database states.

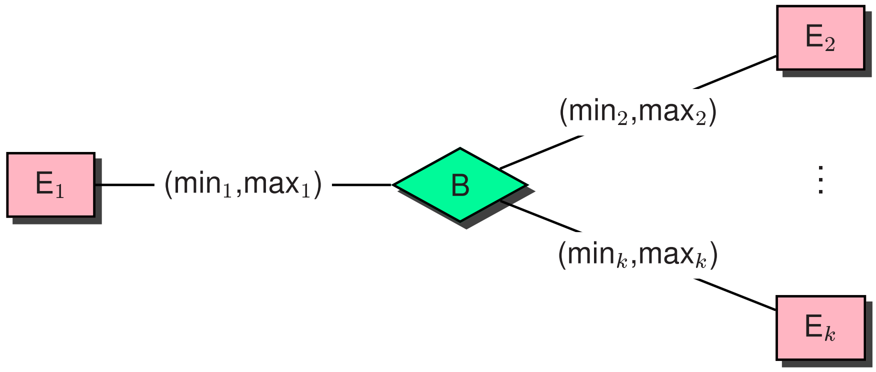
Graphically, key attributes are distinguished by underlining.

Relationship Complexities

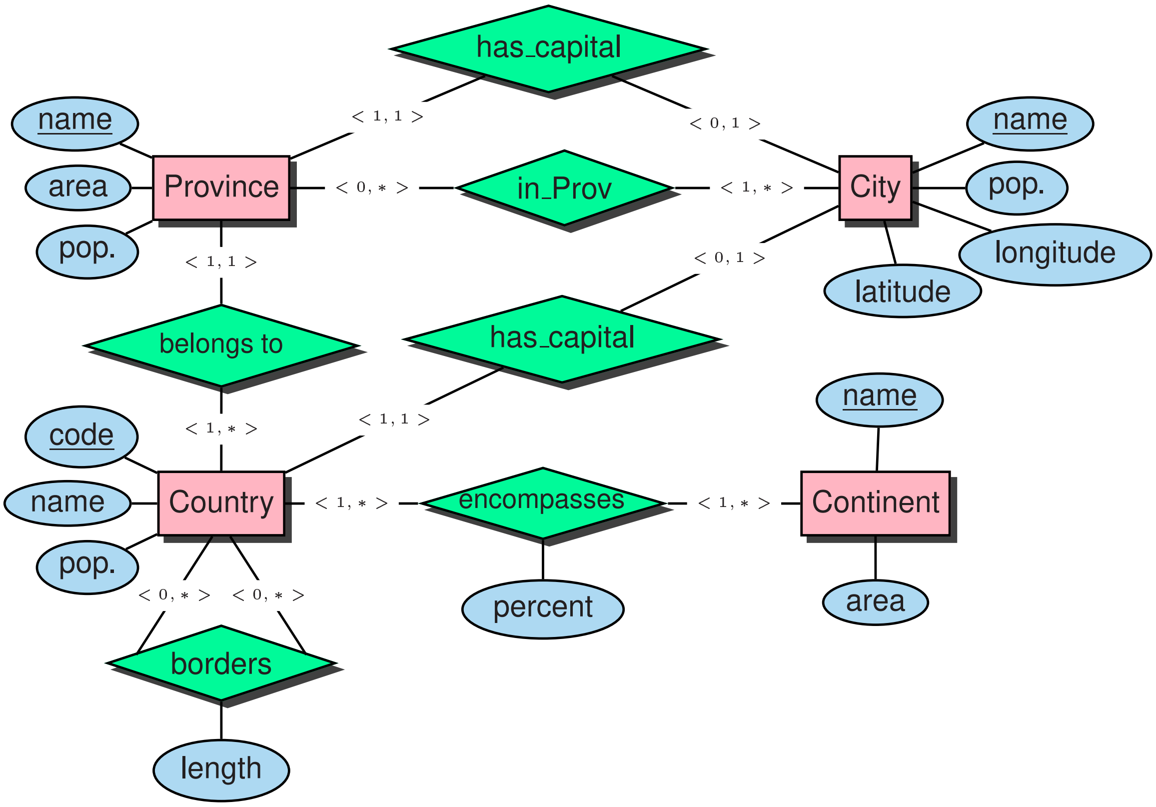
Every relationship type is assigned a complexity that specifies the minimal and maximal number of relationships in which an entity of a given type/role may be involved.

The **complexity degree** of a relationship type B wrt. one of its roles RO is an expression of the form (min, max) where $0 \leq min \leq max$, and $max = *$ means “arbitrary many”.

A set b of relationships of relationship type B satisfies the complexity degree (min, max) of a role RO if for all entities μ of the corresponding entity type E, the following holds: there exist at least min and at most max relationships b in which μ is involved in the role RO.



Example: ER Model of a geographical database



Additional Notions for Complexity Degrees

For *binary* relationships, the following notions are used:

- if $max_1 = max_2 = 1$, it is called a $1 : 1$-relationship.
  is_capital(country:city) is a 1:1-relationship
- if $max_1 > 1, max_2 = 1$, it is called a $n : 1$-relationship (functional relationship) from E_2 to E_1, and a $1 : n$-relationship from E_1 to E_2.
  has_city(country:city) is a 1:n-relationship
- Otherwise, it is called an $n : m$-relationship.
  borders(country:country) is an n:m-relationship

Referential Integrity

Each entity which occurs in a relationship in any database state must also exist in the entity set of this state.

For a relationship type B with relationship set b, a role RO of B that is connected to an entity type E with entity set e, b and e **satisfy the referential integrity** wrt. RO, if for every entity μ that is associated with some $\nu \in b$ under the role RO, $\mu \in e$ holds.

Note:

- referential Integrity is inherent to the ER Model, thus, it is not necessary to care for it.
- there are data models (e.g., the *relational model* (which is described later) where referential integrity must be enforced explicitly).
  (postpone the discussion to the relational model)

2.1.3 Further Concepts

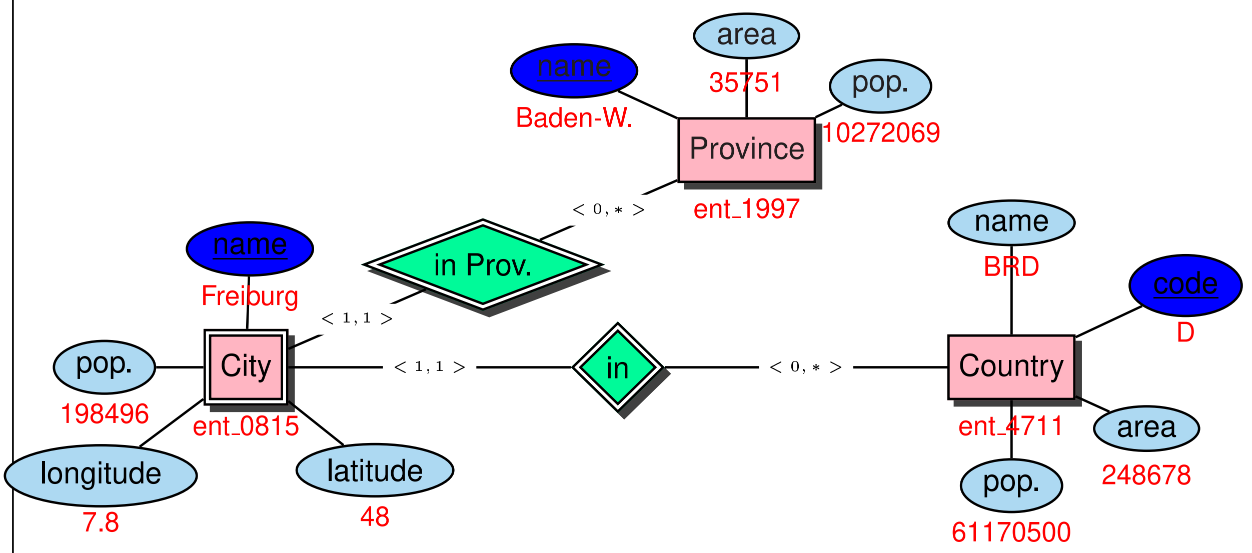
Weak Entity types

A weak entity type is an entity type without a key.

Thus entities of such types must be identified by the help of another entity (see the following figure).

- Weak entity types must be involved in at least one $n : 1$-relationship with a strong entity type (where the strong entity type stands on the 1-side).
  - this relationship is called an *identifying relationship*,
  - the corresponding entity type is called an *identifying entity type*.
- They must have a **local** key, i.e., a set of attributes that can be extended by the primary keys of the corresponding strong entity type to provide a key for the weak entity type (*key inheritance*).
- Note that weak entity types and their identifying relationship types have a special notation.

Weak Entity Types

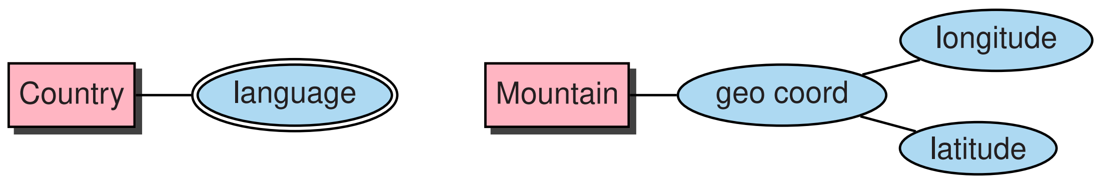


There is also a Freiburg/CH
and Freiburg/Elbe, LowerSaxonia (Niedersachsen)

Extensions of the ERM: Attributes

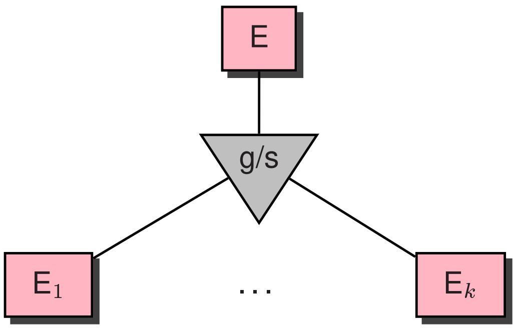
Attributes can be

- set-valued or multi-valued,
- structured



Extensions of the ERM: Generalization/Specialization

Abstractions that allow for grouping entities of different, but related types to a more general type.

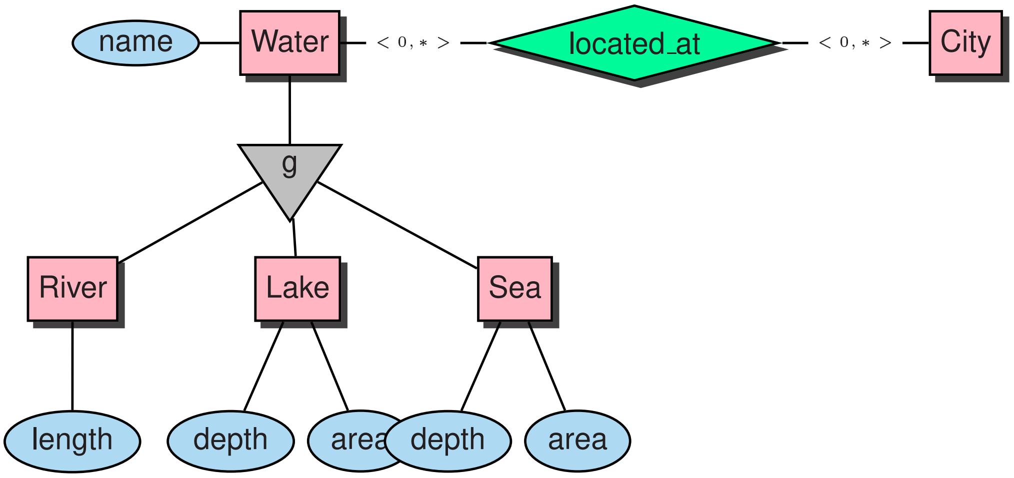


E is called **supertype**, E_i are **subtypes** for $1 \leq i \leq k$. Each entity of a subtype s also an entity of the supertype.

- The common attributes and relationships are assigned to the more general type.
- The attributes and relationships of the supertype are also applicable to the subtypes (which may define further attributes and relationships).

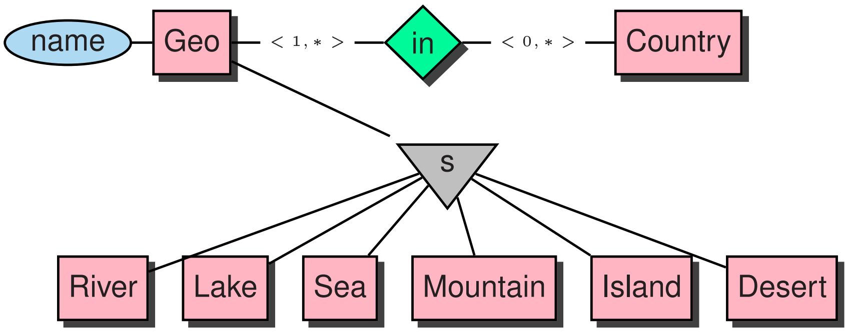
Generalization/Specialization

- Generalization: rivers, lakes, and seas are *waters*. These can e.g. be involved in *located-at* relationships with cities:



GENERALIZATION/SPECIALIZATION

- Specialization: MONDIAL does not describe all geographical things, but only rivers, lakes, seas, mountains, deserts, and islands (no lowlands, highlands, savannas, fens, etc). All such geographical things have in common that they are involved in *in*-relationships with countries:



GENERALIZATION/SPECIALIZATION

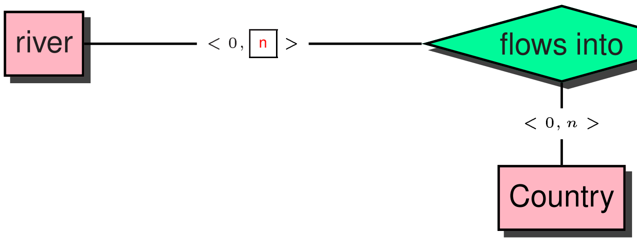
Integrity Constraints

- Common integrity constraints **ISA:** ISA is satisfied in a database state if the entity sets of the subtypes are subsets of the entity sets of the supertype,
- optional integrity constraint **Disjointness:** if the entity sets of the subtypes are disjoint,
- optional integrity constraint **Covering:** if the union of the entity sets of the subtypes cover the entity set of the supertype.

Extensions of the ERM: Aggregation

The ERM does not allow to define relationship types that involve relationship types (note that attributes of relationship types are allowed). This restriction is overcome by defining artificial entity types.

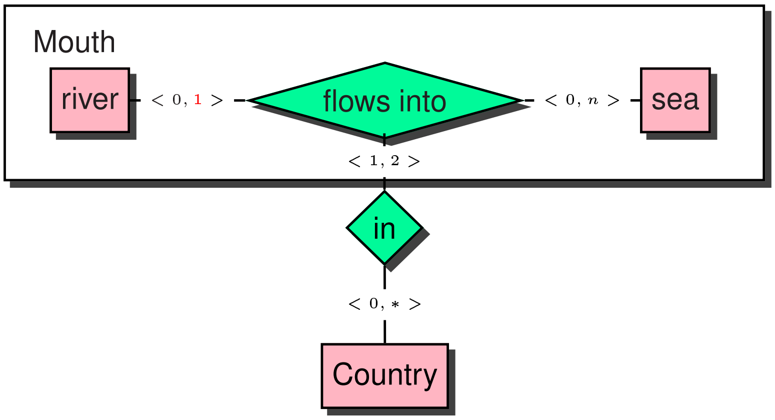
A river flows into a sea/lake/river; more detailed, this point can be described by giving one or two countries.



This representation is ambiguous: A river could flow in two seas!?

Aggregation

Using aggregation, this information can be specified much clearer by introducing an *aggregate type mouth*:



The complexity degrees allow for expressing a more detailed semantics than with the plain ternary relationship type.

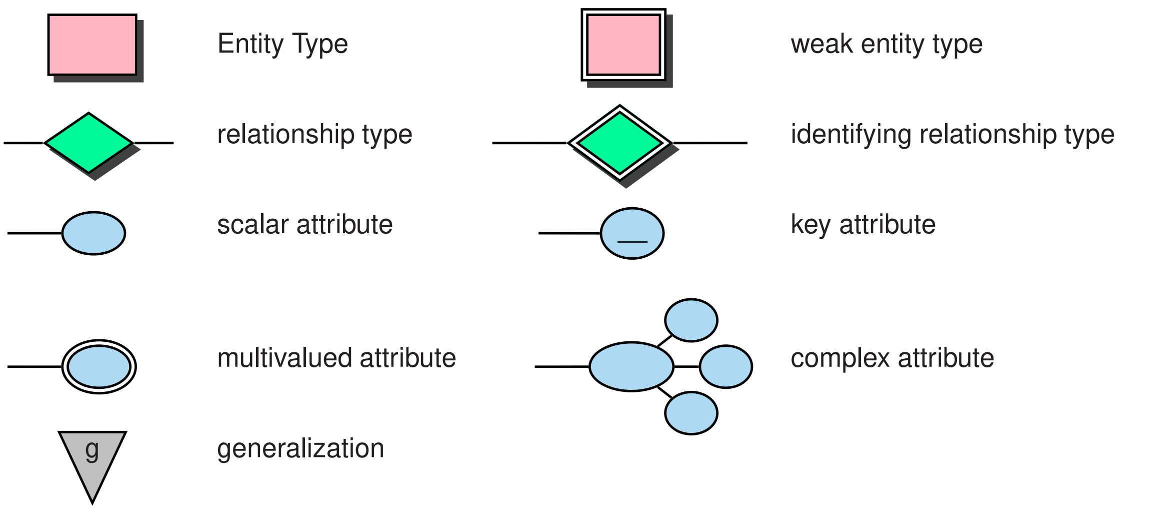
2.1.4 Discussion ERM

- With the structuring concepts of the ERM and its extensions, the *static* aspects of a relevant excerpt of the real world can be modeled semantically adequate in a natural way.
- The graphical representation is also understandable for non-computer-scientists.
- The ERM is useful
  - in the early stages of the design of the database (i.e., when designing the conceptual schema) when discussions with the potential users take place.
  - for documentation (!)
- The ERM can easily be transformed into the data models of existing, real-world database systems (especially, into the relational model – as will be shown in the sequel).
- There are no relevant DBMS that use the ERM directly. They are subsumed by **object-relational** and **object-oriented** DBMS [cf. Lecture on *Information Systems*] (and recently also by XML-based DBMS [cf. Lectures *Databases in the Internet* and *Semistructured Data and XML*]).

Discussion ERM (cont'd)

- There is a more complex and more expressible language:
  UML (Unified Modeling Language)
  [cf. Lecture on *Information Systems*]
  - static aspects are described in more detail than in the ERM, using notions of a fully object-oriented model,
  - dynamic aspects are also described graphically,
  - coarser granularities for describing *information systems* and *workflows* are provided.

Summary: Graphical Notation of ER Constructs



Convention: names of entity types start with a capital letter, names of relationship types and attributes start with non-capital letters.

2.1.5 Some Exercises

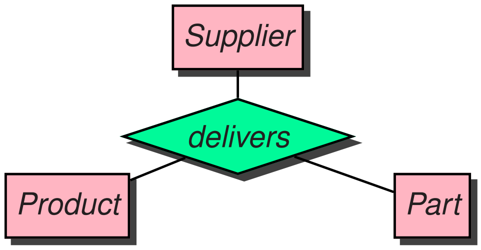
Exercise 2.1
Consider a binary relationship type and the complexities $(0, 1)$ *and* $(1, *)$. *Investigate all possible ways how to assign these relationship complexities to the relationship type. For each variant, give a nontrivial state that satisfies them, and a state that violates them.* □

Exercise 2.2
Discuss ER schemata for the following scenario:

- *All students work on projects. For this, they need tools.* □

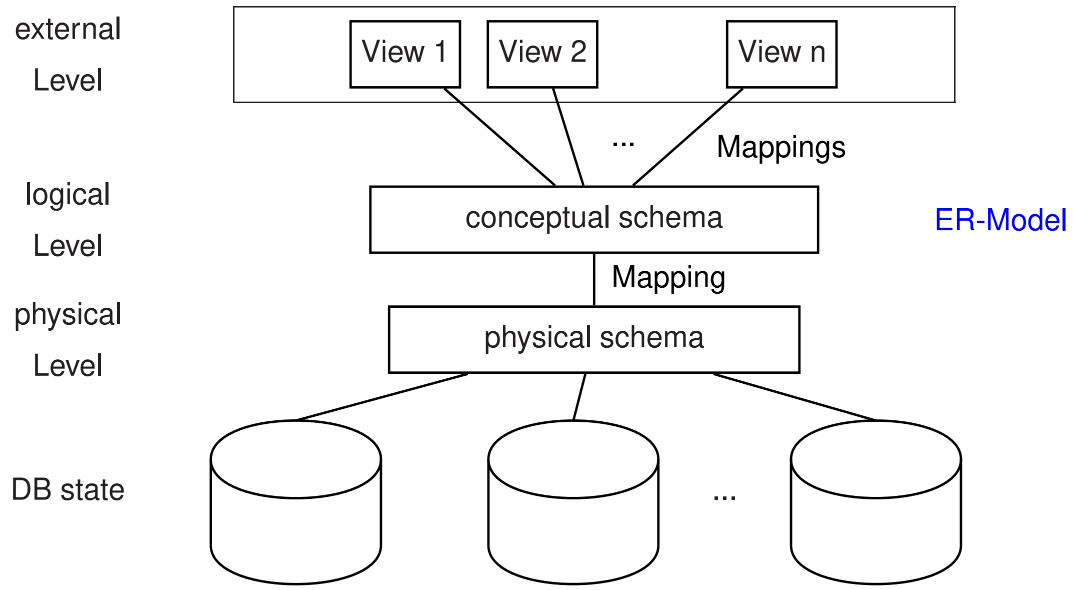
Some Exercises (Cont'd)

Exercise 2.3
Consider a ternary relationship type between the entity types supplier, product, *and* part *(where suppliers deliver parts for a product).*



- *Check whether this situation can be represented by using only binary relationship types.*
  - *Under which conditions is it possible?*
  - *Can such situations be described by the relation complexities?*
- *Show that for an ER schema consisting of a ternary relationship there is always an equivalent ER-Schema that consists of three binary relationship types* and an additional entity type. □

3-Level Architecture of a DBMS (ANSI/SPARC)



2.2 Relational Model (RM)

- *Relational Model* by Codd (1970): mathematical foundation: set theory,
- only a single structural concept *Relation*,
- entity/object types and relationship types are uniformly modeled by **relation schemata**.
- properties of entities/objects and relationships are represented by attributes (in the relation schemata).
- a relation schema consists of a name and a set of attributes,
  Continent: Name, Area
- each attribute is associated with a *domain* that specifies the allowed values of the attribute. Often, attributes also can have a *null value*.
  Continent: Name: VARCHAR(25), Area: NUMBER
- A **(relational) database schema** $\mathbf{R}$ is given by a (finite) set of (relation) schemata.
  Continent: ...; Country: ...; City: ...; encompasses: ...

2.2.1 Relations

- A **(database) state** associates each **relation schema** to a **relation**.
- elements of a relation are called *tuples*.
  Every tuple represents an entity or a relationship. (Name: Asia, area: 4.5E7)
- relations are unordered. Columns are also unordered.

Example:

Continent	
Name	**Area**
VARCHAR(20)	NUMBER
Europe	9562489.6
Africa	3.02547e+07
Asia	4.50953e+07
America	3.9872e+07
Australia	8503474.56

Relations: Example

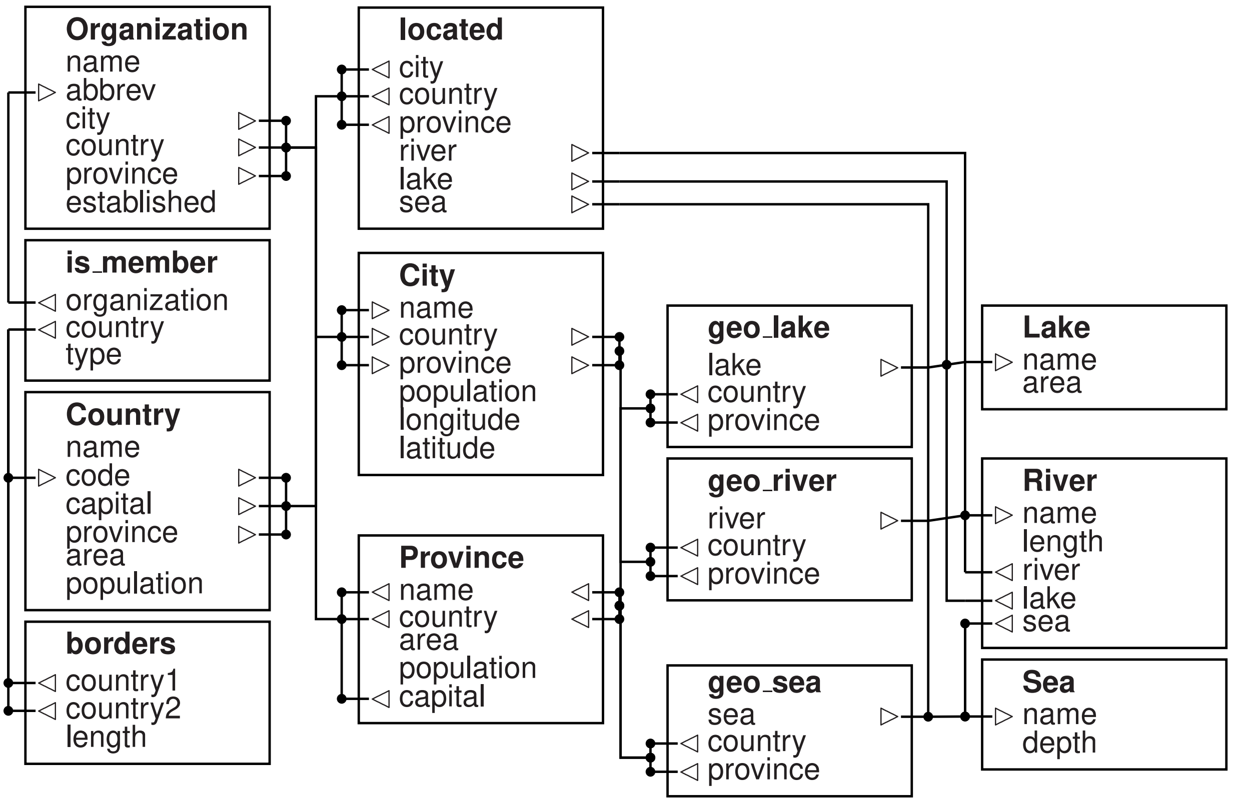
Continent	
Name	**Area**
Europe	9562489.6
Africa	3.02547e+07
Asia	4.50953e+07
America	3.9872e+07
Australia	8503474.56

Country				
Name	**code**	**Population**	**Capital**	**...**
Germany	D	83536115	Berlin	
Sweden	S	8900954	Stockholm	
Canada	CDN	28820671	Ottawa	
Poland	PL	38642565	Warsaw	
Bolivia	BOL	7165257	La Paz	
..	..	..	..	

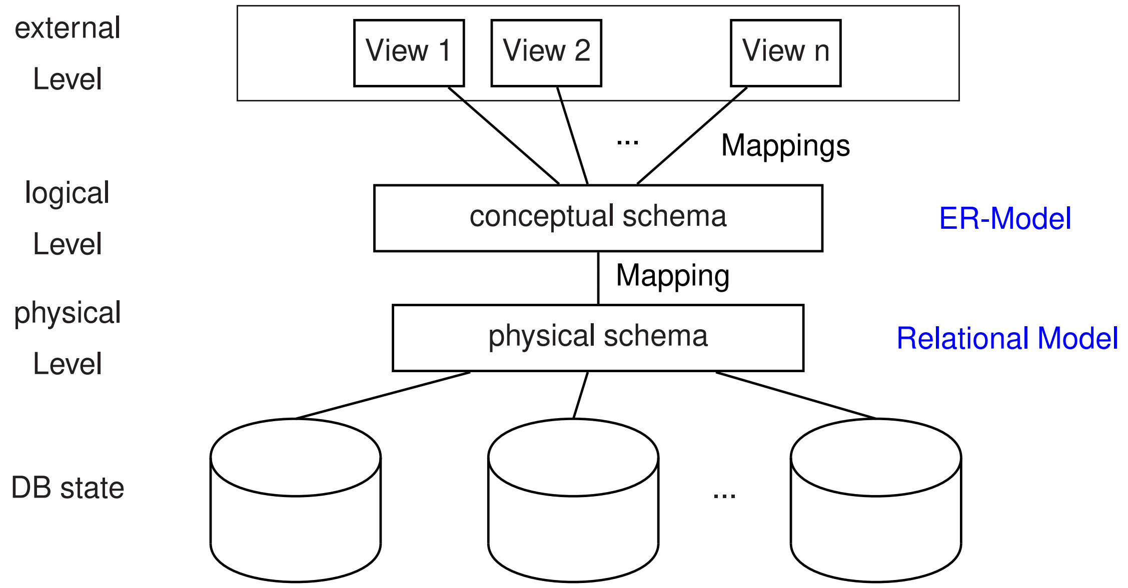
encompasses		
Country	**Continent**	**Percent**
VARCHAR(4)	VARCHAR(20)	NUMBER
R	Europe	20
R	Asia	80
D	Europe	100
...	...	...

- with referential integrity constraints (to be explained later)
- references to keys

Graphical representation of the relational schema of the MONDIAL database (excerpt):



3-Level Architecture of a DBMS (ANSI/SPARC)



2.2.2 Relational Databases – Formalization: Syntax

(note the similarities with first-order logic)

- A **(relational) signature** is a set of **relation schemata** $R_i(\bar{X}_i)$.
- a **relation schema** $R(\bar{X})$ consists of a name (here, R) and a finite set $\bar{X} = \{A_1, \ldots, A_m\}$, $m \geq 1$ of attributes.
  $\bar{X}$ is the **format** of the schema.
- a **(relational) database schema** $\mathbf{R}$ consists of a relational signature (i.e., a set of (relation) schemata), optionally with **integrity constraints**.
- alternative notations for relation schemata:
  - abbreviation: $R(A_1, \ldots, A_n)$ instead of $R(\{A_1, \ldots, A_n\})$.
  - if the order of the attributes $\{A_1, \ldots, A_m\}$ is relevant (i.e., for representation as a table), $\bar{X}$ is denoted as a vector $[A_1, \ldots, A_m]$.

RELATIONAL DATABASES – FORMALIZATION: DOMAINS

Consider a relation schema $R(\bar{X})$

- each $A \in \bar{X}$ is associated to a (non-empty) **domain**, called dom(A).
- dom$(\bar{X})$:= dom$(A_1) \times \ldots \times$ dom(A_m).

Note the following:

- the assignment of domains to attributes belongs to the **database schema**.
- in first-order logic, the definition of the domain of a structure belongs to the **semantics**.

RELATIONAL DATABASES – FORMALIZATION: SEMANTICS

- A **(relational) database** (or, more explicitly, a **database state**) $\mathcal{S}$ (over $\mathbf{R} = \{R_1(\bar{X}_1), \ldots, R_n(\bar{X}_n)\}$) is a **relational structure** over $\mathbf{R}$.
- A **relational structure** $\mathcal{S}$ associates each $R_i(\bar{X}_i)$ to a **relation** $\mathcal{S}(R_i)$ over $\bar{X}_i$.
- elements of a relation are called **tuples**.
  (every tuple represents an entity or a relationship.)
- a **tuple** μ over $\bar{X}$ is a mapping $\mu : \bar{X} \to$ dom$(\bar{X})$; or, for each individual attribute, $\mu : A \to$ dom(A).
  Tup$(\bar{X})$ denotes the set of all tuples over $\bar{X}$.
- a **relation** r over $\bar{X}$ is a finite *set* $r \subseteq$ Tup$(\bar{X})$ – usually represented by a table.
- Rel$(\bar{X})$:= $2^{Tup(\bar{X})}$ is the set of all relations over $\bar{X}$.

PERSPECTIVES: RELATIONAL VS. SET THEORY

- Relations are sets of tuples.
  $\Rightarrow$ **relational algebra**

PERSPECTIVES: RELATIONAL VS. FIRST-ORDER LOGIC

- database schema = relational signature = first-order signature without function symbols
- database = relational structure = first-order structure (without function symbols)
  (some autors use the term "interpretation" instead of "structure")

Relational theory is based on "classical" logic results:
$\Rightarrow$ **relational calculus**

- first-order logic
- finite model theory
- complexity results
- (deductive databases)

KEYS

The notion of **keys** is defined as for the ER model:

For a set $K \subseteq \bar{X}$ of attributes of a relation schema R, a relation $r \in$ Rel$(\bar{X})$ satisfies the **key constraint** K if for all tuples $\mu_1, \mu_2 \in r$:

If $\mu_1(K) = \mu_2(K)$ (i.e., μ_1 and μ_2 coincide in the values of K), then $\mu_1 = \mu_2$.

- In general, there are more than one key (called **candidate keys**) for a relation schema R.
- One of these candidate keys is distinguished (by the designer) to be the **primary key**.
  In the schema, it is represented by underlining these attributes.

Inclusion Constraints and Referential Integrity

Consider relation schemata $R_1(\bar{X}_1)$ und $R_2(\bar{X}_2)$. Let $\bar{Y}_1 \subseteq \bar{X}_1$ and $\bar{Y}_2 \subseteq X_2$ two attribute vectors of the same length.

$r_1 = \mathcal{S}(R_1)$ and $r_2 = \mathcal{S}(R_2)$ satisfy an **inclusion constraint** $R_1.\bar{Y}_1 \subseteq R_2.\bar{Y}_2$ if and only if for all $\mu_1 \in r_1$ there is a $\mu_2 \in r_2$ s.t. $\mu_1(\bar{Y}_1) = \mu_2(\bar{Y}_2)$.

Referential Integrity

- if $\bar{Y}_2$ is the key of R_2, there is a **referential integrity constraint** from $R_1.\bar{Y}_1$ to $R_2.\bar{Y}_2$.
  - Y_1 is called a **foreign key** in R_1 that references $R_2.\bar{Y}_2$.
- encompasses.Continent $\subseteq$ Continent.Name
- encompasses.Country $\subseteq$ Country.Code

... see later where referential integrity constraints come from.

Null Values – Unknown values

- up to now, tuples are **total** functions.
- if for some attribute, there is no value, a **null value** can be used

  Semantics:
  - "value exists, but is unknown"
    (e.g., geo-coordinates of some cities)
  - "value does not yet exist, but will exist in the future"
    (e.g., inflation of a newly founded country)
  - "attribute not applicable"
- a **partial tuple** over $\bar{X}$ is a mapping s.t.

$$(\forall A \in \bar{X})\ \mu(A) \in \text{dom}(A) \cup \{null\}.$$

  A relation is called **partial** if it contains partial tuples.

2.2.3 Exercise

Exercise 2.4
Consider the relation schema $R(\bar{X})$, where $\bar{X} = \{A, B\}$ and dom$(A) =$ dom$(B) = \{1, 2\}$.

- *Give Tup$(\bar{X})$ and Rel$(\bar{X})$.*
- *A is a key of R. Which relations $r \in$ Rel$(\bar{X})$ violate the key constraint?* $\square$

Chapter 3
Database Design

Database design is part of the design phase in the software design process. It is based on the documents of the *requirements analysis*.

3.1 Database Design: Design Steps

Conceptual Design: structuring of the requirements for the representation of the relevant excerpt of the real world:

- independent from the database system to be used (phys. level),
- independent from the detailed views of the users (external schema).

results in the **conceptual schema**, in general an ER schema (or specified in UML).

Implementation Design: Mapping from the conceptual schema to the notions of the database system to be used. The result is the **logical schema**, usually a relational schema (or an object-oriented schema, or – in earlier times – a network database schema).

Detailed Physical Design: definition of the actual storage and appropriate auxiliary data structures (for enhanced efficiency).

"Classical" database design is restricted to the modeling of (static) structures, not considering the (dynamic) processes resulting from the execution (see UML and the *Information Systems* course).

3.2 Logical Schema: Mapping ERM to RM

Starting with the ER schema, the relational schema is designed.

[Overview slide]

Let E_{ER} an entity type and B_{ER} a relationship type in the ERM.

- Entity types: $(E_{ER}, \{A_1, \ldots, A_n\}) \rightarrow E(A_1, \ldots, A_n)$,
- For weak entity types, the key attributes of the identifying entity type must be added.
- Relationship types:
  $(B_{ER}, \{RO_1 : E_1, \ldots, RO_k : E_k\}, \{A_1, \ldots, A_m\}) \rightarrow$
  $B(E_1\_K_{11}, \ldots, E_1\_K_{1p_1},\ \ldots,\ E_k\_K_{k1}, \ldots, E_k\_K_{kp_k},\ A_1, \ldots, A_m)$,
  where $\{K_{i1}, \ldots, K_{ip_i}\}$ are the primary keys of $E_i, 1 \leq i \leq k$.
  - Renaming of foreign key attributes is allowed
    (e.g. coinciding attribute names in different referenced keys)

  In case that $k = 2$ and a (1,1) relationship complexity, the relation schema of the relationship type and that of the entity type may be merged.
- Aggregate types can be ignored if the underlying relationship type is mapped.

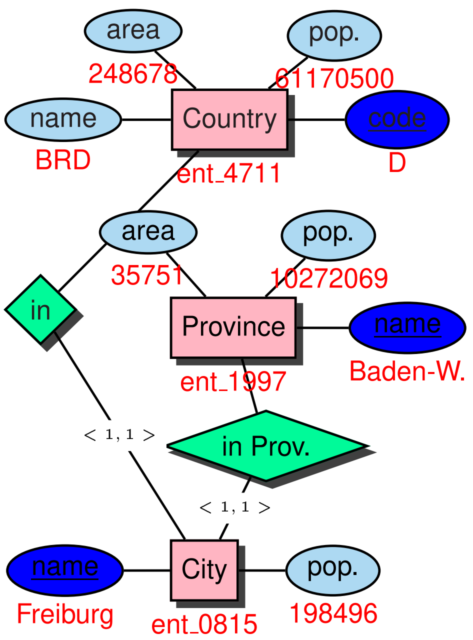
ENTITY TYPES

$(E_{ER}, \{A_1, \ldots, A_n\}) \rightarrow E(A_1, \ldots, A_n)$

Continent	
Name	**Area**
VARCHAR(20)	NUMBER
Europe	9562489.6
Africa	3.02547e+07
Asia	4.50953e+07
America	3.9872e+07
Australia	8503474.56

WEAK ENTITY TYPES

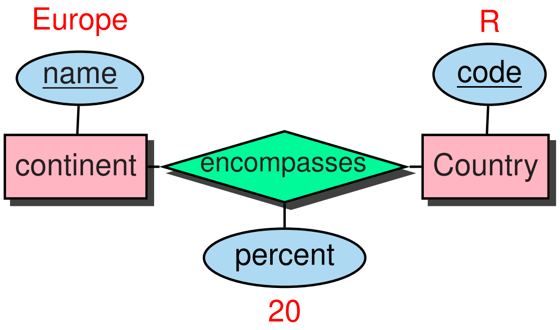
For weak entity types, the key attributes of the identifying entity type(s) must be added.



City				
Name	**Country**	**Province**	**Population**	**...**
Freiburg	D	Baden-W.	198496	..
Berlin	D	Berlin	3472009	..
..	..	..	..	..

Relationship Types

$(B_{ER}, \{RO_1 : E_1, \ldots, RO_k : E_k\}, \{A_1, \ldots, A_m\}) \to$
$B(E_1\_K_{11}, \ldots, E_1\_K_{1p_1}, \ldots, E_k\_K_{k1}, \ldots, E_k\_K_{kp_k},\ A_1, \ldots, A_m)$
where $\{K_{i1}, \ldots, K_{ip_i}\}$ are the primary keys of $E_i, 1 \leq i \leq k$.
(it is allowed to rename, e.g., to use *Country* for *Country.Code*)

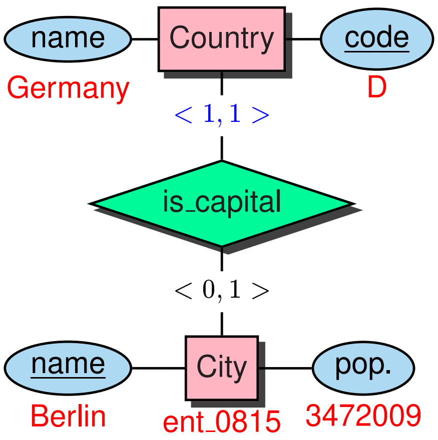


encompasses		
Country	**Continent**	**Percent**
VARCHAR(4)	VARCHAR(20)	NUMBER
R	Europe	20
R	Asia	80
D	Europe	100
...	...	...

- Note: for weak entity types, the global key must be used (exercise: is_capital)

Relationship Types: 1:n-Relationships

In case that $k = 2$ (binary relationship) and a (0,1)- or (1,1)-relationship complexity (i.e., n:1-relations), the relation schema of the relationship type and that of the entity type can be merged (into the relation schema for the entity type)

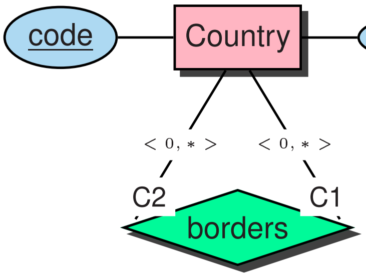


Country					
Name	**code**	**Population**	**Capital**	**Province**	**...**
Germany	D	83536115	Berlin	Berlin	..
Austria	A	8023244	Vienna	Vienna	..
Canada	CDN	28820671	Ottawa	Quebec	..
Bolivia	BOL	7165257	La Paz	Bolivia	..
..	..	..	..	..	..

Other examples: flows_into, headquarters of organizations

RELATIONSHIP TYPES

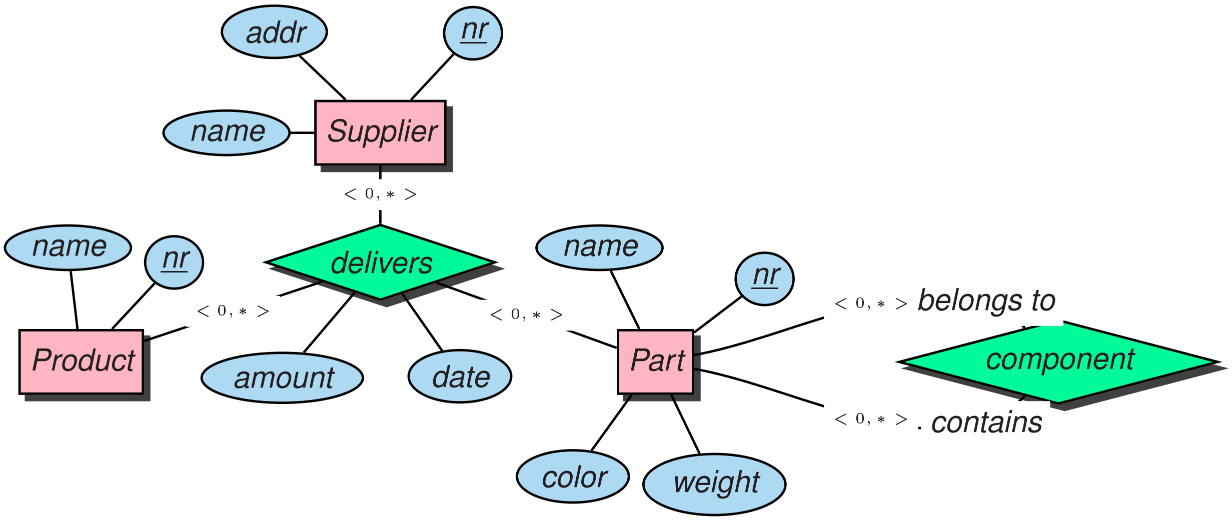
In case that for some relationship type, the keys of involved entity types have coinciding names, the role specifications may be used to guarantee the uniqueness of key attributes in the relationship type.



borders	
Country1	Country2
D	F
D	CH
CH	F
..	..

EXERCISE

Exercise 3.1
Give a relational schema to the following ER schema:



Chapter 4
Relational Database Languages: Relational Algebra

We first consider only *query* languages.

Relational Algebra: Queries are expressions over operators and relation names.

Relational Calculus: Queries are special formulas of first-order logic with free variables.

SQL: Combination from algebra and calculus and additional constructs. Widely used DML for relational databases.

QBE: Graphical query language.

Deductive Databases: Queries are logical rules.

Relational Database Languages: Comparison and Outlook

Remark:

- Relational Algebra and (safe) Relational Calculus have the same expressive power.
  For every expression of the algebra there is an equivalent expression in the calculus, and vice versa.
- A query language is called **relationally complete**, if it is (at least) as expressive as the relational algebra.
- These languages are compromises between efficiency and expressive power; they are not computationally complete (i.e., they cannot simulate a Turing Machine).
- They can be embedded into host languages (e.g. C++ or Java) or extended (PL/SQL), resulting in full computational completeness.
- Deductive Databases (Datalog) are more expressive than relational algebra and calculus.

4.1 Relational Algebra: Computations over Relations

Operations on Tuples – Overview Slide

Let $\mu \in$ Tup$(\bar{X})$ where $\bar{X} = \{A_1, \ldots, A_k\}$.

(Formal definition of μ see Slide 50)

- For $\emptyset \subset \bar{Y} \subseteq \bar{X}$, the expression $\mu[\bar{Y}]$ denotes the **projection** of μ to $\bar{Y}$.

  Result: $\mu[\bar{Y}] \in$ Tup$(\bar{Y})$ where $\mu[\bar{Y}](A) = \mu(A), A \in \bar{Y}$.

- A **selection condition** α (wrt. $\bar{X}$) is an expression of the form $A\,\theta\,B$ or $A\,\theta\,a$, or $a\,\theta\,A$ where $A, B \in \bar{X}$, $a \in$ dom(A), and θ is a comparison operator in $\{=,\neq,\leq,<,\geq,>\}$.

  A tuple $\mu \in$ Tup$(\bar{X})$ **satisfies** a selection condition α, if – according to α – $\mu(A)\,\theta\,\mu(B)$ or $\mu(A)\,\theta\,a$, or $a\,\theta\,\mu(A)$ holds.

  These (atomic) selection conditions can be combined to formulas by using $\wedge$, $\vee$, $\neg$, and $(,)$.

- For $\bar{Y} = \{B_1, \ldots, B_k\}$, the expression $\mu[A_1 \to B_1, \ldots, A_k \to B_k]$ denotes the **renaming** of μ.

  Result: $\mu[\ldots, A_i \to B_i, \ldots] \in$ Tup$(\bar{Y})$ where $\mu[\ldots, A_i \to B_i, \ldots](B_i) = \mu(A_i)$ for $1 \leq i \leq k$.

Let $\mu \in$ Tup$(\bar{X})$ where $\bar{X} = \{A_1, \ldots, A_k\}$.

Projection

For $\emptyset \subset \bar{Y} \subseteq \bar{X}$, the expression $\mu[\bar{Y}]$ denotes the **projection** of μ to $\bar{Y}$.

Result: $\mu[\bar{Y}] \in$ Tup$(\bar{Y})$ where $\mu[\bar{Y}](A) = \mu(A),\ A \in \bar{Y}$.

projection to a given set of attributes

Example 4.1

Consider (for a tuple of the table $R(\bar{X}) = continent(Name, Area)$*):* $\bar{X} = [Name,\ Area]$

and the tuple $\mu =$ $\boxed{\text{“Asia”, 4.50953e+07}}$.

formally: $\mu(Name) =$ *“Asia”*, $\mu(Area) = 4.5E7$

projection attributes: Let $\bar{Y} = [Name]$

Result: $\mu[Name] = \boxed{\text{“Asia”}}$ □

Again, $\mu \in \text{Tup}(\bar{X})$ where $\bar{X} = \{A_1, \ldots, A_k\}$.

Selection

A **selection condition** α (wrt. $\bar{X}$) is an expression of the form $A\,\theta\,B$ or $A\,\theta\,a$, or $a\,\theta\,A$ where $A, B \in \bar{X}$, $a \in \text{dom}(A)$, and θ is a comparison operator in $\{=,\neq,\leq,<,\geq,>\}$.

A tuple $\mu \in \text{Tup}(\bar{X})$ **satisfies** a selection condition α, if – according to α – $\mu[A]\,\theta\,\mu[B]$ or $\mu[A]\,\theta\,a$, or $a\,\theta\,\mu[A]$ holds.

yes/no-selection of tuples (without changing the tuple)

Example 4.2
Consider again (for a tuple of the table $R(\bar{X}) = continent(Name, Area)$): $\bar{X} = [Name,\ Area]$.

Selection condition: $Area > 10.000.000$.

Consider again the tuple $\mu =$ $\boxed{\text{“Asia”, 4.50953e+07}}$.

formally: $\mu(Name) =$ “Asia”, $\mu(Area) = 4.5E7$

check: $\mu(Area) > 10.000.000$

Result: yes. □

These (atomic) selection conditions can be combined to formulas by using $\wedge$, $\vee$, $\neg$, and $(,)$.

Let $\mu \in \text{Tup}(\bar{X})$ where $\bar{X} = \{A_1, \ldots, A_k\}$.

Renaming

For $\bar{Y} = \{B_1, \ldots, B_k\}$, the expression $\mu[A_1 \to B_1, \ldots, A_k \to B_k]$ denotes the **renaming** of μ.

Result: $\mu[\ldots, A_i \to B_i, \ldots] \in \text{Tup}(\bar{Y})$ where $\mu[\ldots, A_i \to B_i, \ldots](B_i) = \mu(A_i)$ for $1 \leq i \leq k$.

renaming of attributes (without changing the tuple)

Example 4.3
Consider (for a tuple of the table $R(\bar{X}) = encompasses(Country, Continent, Percent)$):

$\bar{X} = [Country,\ Continent,\ Percent]$.

Consider the tuple $\mu =$ $\boxed{\text{“R”, “Asia”, 80}}$.

formally: $\mu(Country) =$ “R”, $\mu(Continent) =$ “Asia”, $\mu(Percent) = 80$

Renaming: $\bar{Y} = [Code, Name, Percent]$

Result: a new tuple
$\mu[Country \to Code,\ Continent \to Name,\ Percent \to Percent] =$ $\boxed{\text{“R”, “Asia”, 80}}$ that now fits into the schema $new\_encompasses(Code, Name, Percent)$. □

The usefulness of renaming will become clear later ...

Expressions in the Relational Algebra

What is an algebra?

- An algebra consists of a ”domain” (i.e., a set of ”things”), and a set of operators.
- Operators map elements of the domain to other elements of the domain.
- Each of the operators has a ”semantics”, that is, a definition how the result of applying it to some input should look like.
- **Algebra expressions** are built over basic constants and operators (inductive definition).

Relational Algebra

- The ”domain” consists of all relations (over arbitrary sets of attributes).
- The operators are then used for combining relations, and for describing computations - e.g., in SQL.
- **Relational algebra expressions** are defined inductively over relations and operators.
- Relational algebra expressions define queries against a relational database.

Inductive Definition of Expressions

Atomic Expressions

- For an arbitrary attribute A and a constant $a \in \text{dom}(A)$, the **constant relation** $A : \{a\}$ is an algebra expression.

  Format: $[A]$
  Result relation: $\{a\}$

A:{a}
A
a

- Given a database schema $\mathbf{R} = \{R_1(\bar{X}_1), \ldots, R_n(\bar{X}_n)\}$, every relation name R_i is an algebra expression.

  Format of R_i: $\bar{X}_i$
  Result relation (wrt. a given database state $\mathcal{S}$): the relation $\mathcal{S}(R_i)$ that is currently stored in the database.

Structural Induction: Applying an Operator

- takes one or more input relations $in_1, in_2, \ldots$
- produces a result relation out:
  - out has a **format**, depends on the formats of the input relations.
  - out is a relation, i.e., it contains some tuples, depends on the content of the input relations.

BASE OPERATORS

Let $\bar{X}, \bar{Y}$ formats and $r \in \mathsf{Rel}(\bar{X})$ and $s \in \mathsf{Rel}(\bar{Y})$ relations over $\bar{X}$ and $\bar{Y}$.

Union

Assume $r, s \in \mathsf{Rel}(\bar{X})$.
Result format of $r \cup s$: $\bar{X}$
Result relation: $r \cup s = \{\mu \in \mathsf{Tup}(\bar{X}) \mid \mu \in r \text{ or } \mu \in s\}$.

$r =$

A	B	C
a	b	c
d	a	f
c	b	d

$s =$

A	B	C
b	g	a
d	a	f

$r \cup s =$

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

Set Difference

Assume $r, s \in \text{Rel}(\bar{X})$.
Result format of $r \setminus s$: $\bar{X}$
Result relation: $r \setminus s = \{\mu \in r \mid \mu \notin s\}$.

$$r = \begin{array}{ccc} A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \end{array} \qquad s = \begin{array}{ccc} A & B & C \\ \hline b & g & a \\ d & a & f \end{array} \qquad r \setminus s = \begin{array}{ccc} A & B & C \\ \hline a & b & c \\ c & b & d \end{array}$$

Projection

Assume $r \in \text{Rel}(\bar{X})$ and $\bar{Y} \subseteq \bar{X}$.
Result format of $\pi[\bar{Y}](r)$: $\bar{Y}$
Result relation: $\pi[\bar{Y}](r) = \{\mu[\bar{Y}] \mid \mu \in r\}$.

Example 4.4

Continent	
<u>Name</u>	***Area***
Europe	*9562489.6*
Africa	*3.02547e+07*
Asia	*4.50953e+07*
America	*3.9872e+07*
Australia	*8503474.56*

Let $\bar{Y} = [Name]$

$\mu_1[Name] = \boxed{\text{“Europe”}}$
$\mu_2[Name] = \boxed{\text{“Africa”}}$
$\mu_3[Name] = \boxed{\text{“Asia”}}$
$\mu_4[Name] = \boxed{\text{“America”}}$
$\mu_5[Name] = \boxed{\text{“Australia”}}$

$\pi[Name]$(***Continent***)
Name
Europe
Africa
Asia
America
Australia

□

Selection

Assume $r \in$ Rel$(\bar{X})$ and a selection condition α over $\bar{X}$.

Result format of $\sigma[\alpha](r)$: $\bar{X}$
Result relation: $\sigma[\alpha](r) = \{\mu \in r \mid \mu \text{ satisfies } \alpha\}$.

Example 4.5

Let $\alpha =$ *"$Area > 10.000.000$"*

Continent	
Name	***Area***
Europe	*9562489.6*
Africa	*3.02547e+07*
Asia	*4.50953e+07*
America	*3.9872e+07*
Australia	*8503474.56*

$\mu_1(Area) < 10.000.000$	*no*
$\mu_2(Area) > 10.000.000$	*yes*
$\mu_2(Area) > 10.000.000$	*yes*
$\mu_2(Area) > 10.000.000$	*yes*
$\mu_2(Area) < 10.000.000$	*no*

$\sigma[Area > 10E6]$(***Continent***)	
Name	***Area***
Africa	*3.02547e+07*
Asia	*4.50953e+07*
America	*3.9872e+07*

□

Renaming

Assume $r \in$ Rel$(\bar{X})$ with $X = [A_1, \ldots, A_k]$ and a renaming $[A_1 \to B_1, \ldots, A_k \to B_k]$.

Result format of $\rho[A_1 \to B_1, \ldots, A_k \to B_k](r)$: $[B_1, \ldots, B_k]$
Result relation: $\rho[A_1 \to B_1, \ldots, A_k \to B_k](r) = \{\mu[A_1 \to B_1, \ldots, A_k \to B_k] \mid \mu \in r\}$.

Example 4.6
Consider the renaming of the table $encompasses(Country, Continent, Percent)$*:*

$\bar{X} = [Country,\ Continent,\ Percent]$
Renaming: $\bar{Y} = [Code, Name, Percent]$

$\rho[Country \to Code,\ Continent \to Name,\ Percent \to Percent]$(***encompasses***)		
Code	***Name***	***Percent***
R	*Europe*	*20*
R	*Asia*	*80*
D	*Europe*	*100*
⋮	⋮	⋮

□

(Natural) Join

Assume $r \in \text{Rel}(\bar{X})$ and $s \in \text{Rel}(\bar{Y})$ for arbitrary $\bar{X}, \bar{Y}$.

Convention: Instead of $\bar{X} \cup \bar{Y}$, we also write $\overline{XY}$.

for two tuples $\mu_1 = \boxed{v_1, \ldots, v_n}$ and $\mu_2 = \boxed{w_1, \ldots, w_m}$, $\mu_1\mu_2 := \boxed{v_1, \ldots, v_n, w_1, \ldots, w_m}$.

Result format of $r \bowtie s$: $\overline{XY}$.

Result relation: $r \bowtie s = \{\mu \in \text{Tup}(\overline{XY}) \mid \mu[\bar{X}] \in r \text{ and } \mu[\bar{Y}] \in s\}$.

Motivation

Simplest Case: $\bar{X} \cap \bar{Y} = \emptyset \Rightarrow$ Cartesian Product $r \bowtie s = r \times s$

$r \times s = \{\mu_1\mu_2 \in \text{Tup}(\overline{XY}) \mid \mu_1 \in r \text{ and } \mu_2 \in s\}$.

$r =$

A	B
1	2
4	5

$s =$

C	D
a	b
c	d
e	f

$r \bowtie s =$

A	B	C	D
1	2	a	b
1	2	c	d
1	2	e	f
4	5	a	b
4	5	c	d
4	5	e	f

Example 4.7 (Cartesian Product of Continent and Encompasses)

Continent $\times$ ***encompasses***				
Name	***Area***	***Continent***	***Country***	***Percent***
Europe	*9562489.6*	*Europe*	*Germany*	*100*
Europe	*9562489.6*	*Europe*	*Russia*	*20*
Europe	*9562489.6*	*Asia*	*Russia*	*80*
Europe	*9562489.6*	*:*	*:*	*:*
Africa	*3.02547e+07*	*Europe*	*Germany*	*100*
Africa	*3.02547e+07*	*Europe*	*Russia*	*20*
Africa	*3.02547e+07*	*Asia*	*Russia*	*80*
Africa	*3.02547e+07*	*:*	*:*	*:*
Asia	*4.50953e+07*	*Europe*	*Germany*	*100*
Asia	*4.50953e+07*	*Europe*	*Russia*	*20*
Asia	*4.50953e+07*	*Asia*	*Russia*	*80*
Asia	*4.50953e+07*	*:*	*:*	*:*
:	*:*	*:*	*:*	*:*

□

Back to the Natural Join

General Case $\bar{X} \cap \bar{Y} \neq \emptyset$: shared attribute names constrain the result relation.

Again the definition: $r \bowtie s = \{\mu \in \text{Tup}(\overline{XY}) \mid \mu[\bar{X}] \in r \text{ and } \mu[\bar{Y}] \in s\}$.

Example 4.8

Consider encompasses(country,continent,percentage) and is_member(organization,country,type):

encompasses		
Country	***Continent***	***Percent***
R	*Europe*	*20*
R	*Asia*	*80*
D	*Europe*	*100*
:	*:*	*:*

is_member		
Organization	***Country***	***Type***
EU	*D*	*member*
UN	*D*	*member*
UN	*R*	*member*
:	*:*	*:*

$$encompasses \bowtie is\_member = \{\mu \in \text{Tup}(country, cont, perc, org, type) \mid \mu[country, cont, perc] \in encompasses \text{ and } \mu[org, country, type] \in is\_member\}$$

□

Example 4.8 (Continued)

$$encompasses \bowtie is\_member = \{\mu \in \text{Tup}(country, cont, perc, org, type) \mid \mu[country, cont, perc] \in encompasses \text{ and } \mu[org, country, type] \in is\_member\}$$

start with $(R, Europe, 20) \in encompasses.$
check which tuples in is_member *match:*
- $(UN, R, member) \in is\_member$ *matches:*
  result: $(R, Europe, 20, UN, member)$ *belongs to the result.*
  (some more matches ...)

continue with $(R, Asia, 80) \in encompasses.$
- $(UN, R, member) \in is\_member$ *matches:*
  result: $(R, Asia, 80, UN, member)$ *belongs to the result.*
  (some more matches ...)

continue with $(D, Europe, 100) \in encompasses.$
- $(EU, D, member) \in is\_member$ *matches:*
  result: $(D, Europe, 100, EU, member)$ *belongs to the result.*
- $(UN, D, member) \in is\_member$ *matches:*
  result: $(D, Europe, 100, UN, member)$ *belongs to the result.*
  (some more matches ...)

Example 4.8 (Continued)
Result:

encompasses $\times$ ***is_member***				
Country	***Continent***	***Percent***	***Organization***	***Type***
R	*Europe*	*20*	*UN*	*member*
R	*Europe*	*20*	*:*	*:*
R	*Asia*	*80*	*UN*	*member*
R	*Asia*	*80*	*:*	*:*
D	*Europe*	*100*	*UN*	*member*
D	*Europe*	*100*	*EU*	*member*
D	*Europe*	*100*	*:*	*:*
:	*:*	*:*	*:*	*:*

□

Example 4.9 (and Exercise)
Consider the expression

$$continent \bowtie \rho[Country \to Code,\ Continent \to Name,\ Percent \to Percent](\textbf{\textit{encompasses}})$$

□

Derived Operators

Intersection

Assume $r, s \in \text{Rel}(\bar{X})$.

Then, $r \cap s = \{\mu \in \text{Tup}(\bar{X}) \mid \mu \in r \text{ and } \mu \in s\}$.

Theorem 4.1
Intersection can be expressed by Difference: $r \cap s = r \setminus (r \setminus s)$. □

Relational Division

Assume $r \in \text{Rel}(\bar{X})$ and $s \in \text{Rel}(\bar{Y})$ such that $\bar{Y} \subsetneq \bar{X}$.
Result format of $r \div s$: $\bar{Z} = \bar{X} \setminus \bar{Y}$.

The result relation $r \div s$ is specified as "all $\bar{Z}$-values that occur in $\pi(\bar{Z})(r)$, with the additional condition that they occur in r together with each of the $\bar{Y}$ values that occur in s".

Formally,

$$r \div s = \{\mu \in \text{Tup}(Z) \mid \{\mu\} \times s \subseteq r\} = \pi[Z]r \setminus \pi[Z]((\pi[Z]r \times s) \setminus r).$$

this implies that $\mu \in \pi(\bar{Z})(r)$

- Simple observation: $\pi(\bar{Z})(r) \supseteq r \div s$.
  This constrains the set of possible results.

Example 4.10 (Relational Division)
Compute all countries that belong both to Europe and to Asia:

enc	
country	***continent***
R	*Asia*
R	*Europe*
IND	*Asia*
D	*Europe*
TR	*Asia*
TR	*Europe*
ET	*Africa*
ET	*Asia*
CH	*Europe*
:	*:*

cts
continent
Asia
Europe

Compute $enc \div cts$*:*
$\bar{X} = [country, continent]$, $\bar{Y} = [continent]$
Thus, $\bar{Z} = [country]$.
Consider all values in $\pi[country](enc)$*:*
Start with "R" $\in \pi[country](enc)$*:*
for "Asia" $\in cts$*: ("R","Asia")* $\in enc$.
for "Europe" $\in cts$*: ("R","Europe")* $\in enc$.
OK. "R" belongs to the result.
Continue with "IND" $\in \pi[country](enc)$*:*
for "Asia" $\in cts$*: ("IND","Asia")* $\in enc$.
for "Europe" $\in cts$*: ("IND","Europe")* $\notin enc$.
"IND" does not belong to the result.
:
"TR" belongs to the result.
"ET" does not belong to the result.
"CH" does not belong to the result.

□

Example 4.10 (Cont(d))
Consider again Example 4.10 and the formal algebraic characterization of Division:

$$r \div s = \{\mu \in \mathit{Tup}(Z) \mid \{\mu\} \times s \subseteq r\} = \pi[Z]r \setminus \pi[Z]((\pi[Z]r \times s) \setminus r).$$

1. $r = belongs\_to$, $s = continent$, $Z = Country$.
2. $(\pi[Z]r \times s)$ *contains all tuples of countries with Europe and Asia, e.g., (Germany,Europe), (Germany,Asia), (Russia,Europe), (Russia,Asia)*
3. $((\pi[Z]r \times s) \setminus r)$ *contains all sich tuples which are not "valid", e.g., (Germany,Asia).*
4. *projecting this to the countries yields all those countries where a non-valid tuple has been generated in (2), i.e., which do not belong both to Europe and Asia.*
5. $\pi[Z]r$ *is the list of all countries ...*
6. *... subtracting those computed in (4) yields those that belong both to Europe and Asia.* □

θ-Join

Combination of Cartesian Product and Selection:

Assume $r \in \mathrm{Rel}(\bar{X})$, and $s \in \mathrm{Rel}(\bar{Y})$, such that $\bar{X} \cap \bar{Y} = \emptyset$, and $A\,\theta\,B$ a selection condition.

$$r \bowtie_{A\theta B} s = \{\mu \in \mathrm{Tup}(\overline{XY}) \mid \mu[\bar{X}] \in r,\ \mu[\bar{Y}] \in s \text{ and } \mu \text{ satisfies } A\theta B\} = \sigma[A\theta B](r \times s).$$

Equi-Join

θ-join that uses the "="-predicate.

Example 4.11 (and Exercise)
Consider again Example 4.7:

$Continent \times encompasses$ *contained tuples that did not really make sense.*

$(Continent \times encompasses)_{continent=name}$ *would be more useful.*

Furthermore, consider
$\pi[continent, area, code, percent]((Continent \times encompasses)_{continent=name})$:

- *removes the - now redundant - "name" column,*
- *is equivalent to the natural join* $(\rho[name \to continent]continent) \bowtie encompasses.$ □

SEVERAL EXTENSIONS OF THE JOIN

- Join is the operator for combining relations

Example 4.12
Consider a completely different database now for investigating joins.

- *Persons work in divisions of a company*
- *Tools are assigned to the divisions*

Works

Person	***Division***
John	*Production*
Bill	*Production*
John	*Research*
Mary	*Research*
Sue	*Sales*

Tools

Division	***Tool***
Production	*hammer*
Research	*pen*
Research	*computer*
Administration	*typewriter*

□

Example 4.12 (Continued)
Consider the join of both tables:

Works

Person	***Division***
John	*Production*
Bill	*Production*
John	*Research*
Mary	*Research*
Sue	*Sales*

Tools

Division	***Tool***
Production	*hammer*
Research	*pen*
Research	*computer*
Admin.	*typewriter*

Works $\bowtie$ Tools

Person	***Division***	***Tool***
John	*Production*	*hammer*
Bill	*Production*	*hammer*
John	*Research*	*pen*
John	*Research*	*computer*
Mary	*Research*	*pen*
Mary	*Research*	*computer*

- *there is no tuple that describes Sue*
- *there is no tuple that describes the administration or sales division*
- *there is no tuple that shows that there is a typewriter*

□

Semi-Join

Assume $r \in \text{Rel}(\bar{X})$ and $s \in \text{Rel}(\bar{Y})$ such that $\bar{X} \cap \bar{Y} \neq \emptyset$.

Result format: $\bar{X}$
Result relation: $r \ltimes s = \pi[\bar{X}](r \bowtie s)$

The semi-join $r \ltimes s$ does *not* return the join, but checks which tuples of r "survive" the join with s (i.e., "which find a counterpart in s wrt. the shared attributes"):

Example 4.13
Consider again Example 4.12:

Works $\ltimes$ Tools	
Person	***Division***
John	*Production*
Bill	*Production*
John	*Research*
Mary	*Research*

Works $\rtimes$ Tools	
Division	***Tool***
Production	*hammer*
Research	*pen*
Research	*computer*

□

- Semijoins are e.g. used for optimizing the evaluation of multiple joins.
  [see lecture on Database Theory]

Outer Join

Assume $r \in \text{Rel}(\bar{X})$ and $s \in \text{Rel}(\bar{Y})$.

Result format of r ⟗ s: $\overline{XY}$
The outer join extends the "inner" join with all tuples that have no counterpart in the other relation (filled with null values):

Example 4.14 (Outer Join)
Consider again Example 4.12

Works ⟗ Tools		
Person	***Division***	***Tool***
John	*Production*	*hammer*
Bill	*Production*	*hammer*
John	*Research*	*pen*
John	*Research*	*computer*
Mary	*Research*	*pen*
Mary	*Research*	*computer*
Sue	*Sales*	*NULL*
NULL	*Admin*	*typewriter*

□

Formally, the result relation is defined as follows:

$J = r \bowtie s$ — take the ("inner") join as base
$r_0 = r \setminus \pi[\bar{X}]J = r \setminus r \ltimes s$ — r-tuples that "are missing"
$s_0 = s \setminus \pi[\bar{Y}]J = s \setminus r \rtimes s$ — s-tuples that "are missing"
$Y_0 = \bar{Y} \setminus \bar{X}$, $X_0 = \bar{X} \setminus \bar{Y}$
Let $\mu_1 \in \mathrm{Tup}(Y_0)$, $\mu_2 \in \mathrm{Tup}(X_0)$ such that μ_1, μ_2 consist only of *null* values

$$r ⟗ s = J \cup (r_0 \times \{\mu_1\}) \cup (s_0 \times \{\mu_2\}).$$

Example 4.14 (Continued)
For the above example,

$J = Works \bowtie Tools$
$r_0 = [$*"Sue","Sales"*$]$, $s_0 = [$*"Admin","Typewriter"*$]$
$Y_0 = Tool$, $X_0 = Person$

$\mu_1 =$

Tool
$null$

$\mu_2 =$

Person
$null$

$r_0 \times \{\mu_1\} =$

Person	**Division**	**Tool**
Sue	$Sales$	$null$

$s_0 \times \{\mu_2\} =$

Person	**Division**	**Tool**
$null$	$Admin$	$Typewriter$

□

Generalized Natural Join

Assume $r_i \subseteq \mathrm{Tup}(\bar{X}_i)$.

Result format: $\cup_{i=1}^{n} \bar{X}_i$
Result relation: $\bowtie_{i=1}^{n} r_i = \{\mu \in \mathrm{Tup}(\cup_{i=1}^{n} \bar{X}_i) \mid \mu[\bar{X}_i] \in r_i\}$

Exercise 4.1
Prove that the natural join is commutative (which makes the generalized natural join well-defined):

$$\begin{aligned} \bowtie_{i=1}^{n} r_i &= ((\ldots((r_1 \bowtie r_2) \bowtie r_3) \bowtie \ldots) \bowtie r_n) \\ &= (r_1 \bowtie (r_2 \ldots (r_{n-1} \bowtie r_n) \ldots)) \end{aligned}$$

□

Expressions

- inductively defined: combining expressions by operators

Example 4.15
The names of all cities where (i) headquarters of an organization are located, and (ii) that are capitals of a member country of this organization.

As a tree:

$$\pi[City]$$

$$\cap$$

$\pi[Abbrev, City, Prov, Country]$ $\qquad$ $\rho[Capital \to City]$

Organization $\qquad$ $\pi[Abbrev, Capital, Prov, Country]$

$$\bowtie$$

$\rho[Organization \to Abbrev]$ $\qquad$ $\rho[Code \to Country]$

is_Member $\qquad$ *Country* $\qquad \square$

Note that there are many equivalent expressions.

Expressions in the Relational Algebra as Queries

Let $\mathbf{R} = \{R_1, \ldots, R_k\}$ a set of relation schemata of the form $R_i(\bar{X}_i)$. As already described, an **database state** to $\mathbf{R}$ is a **structure** $\mathcal{S}$ that maps every relation name R_i in $\mathbf{R}$ to a relation $\mathcal{S}(R_i) \subseteq$ Tup$(\bar{X}_i)$

Every algebra expression Q defines a **query** against the state $\mathcal{S}$ of the database:

- For given $\mathbf{R}$, Q is assigned a **format** Σ_Q (the format of the answer).
- For every database state $\mathcal{S}$, $\mathcal{S}(Q) \subseteq$ Tup(Σ_Q) is a relation over Σ_Q, called the **answer set** for Q wrt. $\mathcal{S}$.
- $\mathcal{S}(Q)$ can be computed according to the inductive definition, starting with the innermost (atomic) subexpressions.
- Thus, the relational algebra has a **functional semantics**.

Summary: Inductive Definition of Expressions

Atomic Expressions

- For an arbitrary attribute A and a constant $a \in \text{dom}(A)$, the **constant relation** $A : \{a\}$ is an algebra expression.

  $\Sigma_{A:\{a\}} = [A]$ and $\mathcal{S}(A : \{a\}) = A : \{a\}$
- Every relation name R is an algebra expression.

  $\Sigma_R = \bar{X}$ and $\mathcal{S}(R) = \mathcal{S}(R)$.

Summary (Cont'd)

Compound Expressions

Assume algebra expressions Q_1, Q_2 that define Σ_{Q_1}, Σ_{Q_2}, $\mathcal{S}(Q_1)$, and $\mathcal{S}(Q_2)$.

Compound algebraic expressions are now formed by the following rules (corresponding to the algebra operators):

Union

If $\Sigma_{Q_1} = \Sigma_{Q_2}$, then $Q = (Q_1 \cup Q_2)$ is the **union** of Q_1 and Q_2.

$\Sigma_Q = \Sigma_{Q_1}$ and $\mathcal{S}(Q) = \mathcal{S}(Q_1) \cup \mathcal{S}(Q_2)$.

Difference

If $\Sigma_{Q_1} = \Sigma_{Q_2}$, then $Q = (Q_1 - Q_2)$ is the **difference** of Q_1 and Q_2.

$\Sigma_Q = \Sigma_{Q_1}$ and $\mathcal{S}(Q) = \mathcal{S}(Q_1) \setminus \mathcal{S}(Q_2)$.

Projection

For $\emptyset \neq \bar{Y} \subseteq \Sigma_{Q_1}$, $Q = \pi[\bar{Y}]Q_1$ is the **projection** of Q_1 to the attributes in $\bar{Y}$.

$\Sigma_Q = \bar{Y}$ und $\mathcal{S}(Q) = \pi[\bar{Y}](\mathcal{S}(Q_1))$.

INDUCTIVE DEFINITION OF EXPRESSIONS (CONT'D)

Selection

For a selection condition α over Σ_{Q_1}, $Q = \sigma[\alpha]Q_1$ is the **selection** from Q_1 wrt. α.

$\Sigma_Q = \Sigma_{Q_1}$ and $\mathcal{S}(Q) = \sigma[\alpha](\mathcal{S}(Q_1))$.

Natural Join

$Q = (Q_1 \bowtie Q_2)$ is the **(natural) join** of Q_1 and Q_2.

$\Sigma_Q = \Sigma_{Q_1} \cup \Sigma_{Q_2}$ and $\mathcal{S}(Q) = \mathcal{S}(Q_1) \bowtie \mathcal{S}(Q_2)$.

Renaming

For $\Sigma_{Q_1} = \{A_1, \ldots, A_k\}$ and $\{B_1, \ldots, B_k\}$ a set of attributes, $\rho[A_1 \to B_1, \ldots, A_k \to B_k]Q_1$ is the **renaming** of Q_1

$\Sigma_Q = \{B_1, \ldots, B_k\}$ and $\mathcal{S}(Q) = \{\mu[A_1 \to B_1, \ldots, A_k \to B_k] \mid \mu \in \mathcal{S}(Q_1)\}$.

Example

Example 4.16
Professor(PNr, Name, Office), Course(CNr, Credits, CName)
teach(PNr, CNr), examine(PNr, CNr)

- *For each professor (name) determine the courses he gives (CName).*

  $$\pi\ [\text{Name, CName}]\ ((\text{Professor} \bowtie \text{teach}) \bowtie \text{Course})$$

- *For each professor (name) determine the courses (CName) that he teaches, but that he does not examine.*

  $$\begin{array}{l} \pi[\mathsf{Name}, \mathsf{CName}]((\\ (\pi[\mathsf{Name}, \mathsf{CNr}](\mathsf{Professor} \bowtie \mathsf{teach})) \\ \setminus \\ (\pi[\mathsf{Name}, \mathsf{CNr}](\mathsf{Professor} \bowtie \mathsf{examine})) \\ ) \bowtie \mathsf{Course}) \end{array}$$

□

Equivalence of Expressions

Algebra expressions Q, Q' are called **equivalent**, $Q \equiv Q'$, if and only if for all structures $\mathcal{S}$, $\mathcal{S}(Q) = \mathcal{S}(Q')$.

Equivalence of expressions is the basis for **algebraic optimization**.

Let attr(α) the set of attributes that occur in a selection condition α, and $Q, Q_1, Q_2, \ldots$ expressions with formats X, X_1, $\ldots$.

Projections

- $\bar{Z}, \bar{Y} \subseteq \bar{X} \Rightarrow \pi[\bar{Z}](\pi[\bar{Y}](Q)) \equiv \pi[\bar{Z} \cap \bar{Y}](Q)$.
- $\bar{Z} \subseteq \bar{Y} \subseteq \bar{X} \Rightarrow \pi[\bar{Z}](\pi[\bar{Y}](Q)) \equiv \pi[\bar{Z}](Q)$.

Selections

- $\sigma[\alpha_1](\sigma[\alpha_2](Q)) \equiv \sigma[\alpha_2](\sigma[\alpha_1](Q)) \equiv \sigma[\alpha_1 \wedge \alpha_2](Q))$.
- attr$(\alpha) \subseteq \bar{Y} \subseteq \bar{X} \Rightarrow \pi[\bar{Y}](\sigma[\alpha](Q)) \equiv \sigma[\alpha](\pi[\bar{Y}](Q))$.

Joins

- $Q_1 \bowtie Q_2 \equiv Q_2 \bowtie Q_1$.
- $(Q_1 \bowtie Q_2) \bowtie Q_3 \equiv Q_1 \bowtie (Q_2 \bowtie Q_3)$.

Equivalence of Expressions (Cont'd)

Joins and other Operations

- attr$(\alpha) \subseteq \bar{X}_1 \cap \bar{X}_2 \Rightarrow \sigma[\alpha](Q_1 \bowtie Q_2) \equiv \sigma[\alpha](Q_1) \bowtie \sigma[\alpha](Q_2)$.
- attr$(\alpha) \subseteq \bar{X}_1$, attr$(\alpha) \cap \bar{X}_2 = \emptyset \Rightarrow \sigma[\alpha](Q_1 \bowtie Q_2) \equiv \sigma[\alpha](Q_1) \bowtie Q_2$.
- Assume $V \subseteq \overline{X_1 X_2}$ and let $W = \bar{X}_1 \cap \overline{V X_2}$, $U = \bar{X}_2 \cap \overline{V X_1}$.
  Then, $\pi[V](Q_1 \bowtie Q_2) = \pi[V](\pi[W](Q_1) \bowtie \pi[U](Q_2))$;
- $\bar{X}_2 = \bar{X}_3 \Rightarrow Q_1 \bowtie (Q_2 \text{ op } Q_3) = (Q_1 \bowtie Q_2) \text{ op } (Q_1 \bowtie Q_3)$ where op $\in \{\cup, -\}$.

Exercise 4.2
Prove some of the equalities (use the definitions given on the "Base Operators" slide). □

EXPRESSIVE POWER OF THE ALGEBRA

Theorem 4.2
There is no expression of the relational algebra that computes the transitive closure of arbitrary binary relations r*.* □

EXAMPLES

Time to play. Perhaps postpone examples after comparison with SQL (next subsections)

4.2 SQL

SQL: Structured (Standard) Query Language

Literature: A Guide to the SQL Standard, 3rd Edition, C.J. Date and H. Darwen, Addison-Wesley 1993

History: about 1974 as SEQUEL (IBM, System R)

Standardization:

SQL-86 and **SQL-89:** core language, based on existing implementations, including procedural extensions

SQL-92 (SQL2): some additions

SQL-99 (SQL3):

- active rules
- recursion
- objekt-relational and object-oriented concepts

Underlying Data Model

SQL uses the relational model:

- SQL relations are **multisets (bags)** of tuples (i.e., they can contain duplicates)
- Notions: Relation $\rightsquigarrow$ Table
  Tuple $\rightsquigarrow$ Row
  Attribute $\rightsquigarrow$ Column

The relational algebra serves as theoretical base for SQL as a query language.

The Language

Basic structure of SQL queries:

SELECT $A_1, \ldots, A_n$	(...corresponds to π in the algebra)
FROM $R_1, \ldots, R_m$	(...specifies the contributing relations)
WHERE F	(...corresponds to σ in the algebra)

corresponds to the algebra expression $\pi[A_1, \ldots, A_n](\sigma[F](r_1 \times \ldots \times r_m))$

- Note: cartesian product $\rightarrow$ aliasing

General Structure:

SELECT $A_1, \ldots, A_n$	list of attributes
FROM $R_1, \ldots, R_m$	list of relations
WHERE F	condition(s)
GROUP BY $B_1, \ldots, B_k$	list of grouping attributes
HAVING G	condition on groups
ORDER BY H	sort order

Subqueries correspond to subtrees in the relational algebra.

Constructing Queries

For each problem there are multiple possible equivalent queries in SQL (cf. Example 4.15). The choice is a matter of personal taste.

- analyze the problem "systematically":
  - collect all relations (in the FROM clause) that are needed
  - generate a suitable conjunctive WHERE clause

  $\Rightarrow$ leads to a single "broad" SFW query
  (cf. conjunctive queries, relational calculus)

- analyze the problem "top-down":
  - take the relations that directly contribute to the result in the (outer) FROM clause
  - do all further work in correlated subquery/-queries in the WHERE clause

  $\Rightarrow$ leads to a "main" part and nested subproblems

- decomposition of the problem into subproblems:
  - subproblems are solved by nested SFW queries that are combined in the FROM clause of a surrounding query

Comparison

SQL:

SELECT $A_1, \ldots, A_n$ FROM R_1 AS t_1, . . . ,R_m AS t_m WHERE F

- **equivalent expression in the relational algebra:**

$$\pi[A_1, \ldots, A_n](\sigma[F](r_1 \times \ldots \times r_m))$$

- **Algorithm (nested-loop):**
  FOR each tuple t_1 in relation R_1 DO
      FOR each tuple t_2 in relation R_2 DO
          :
              FOR each tuple t_n in relation R_n DO
                  IF WHERE-clause is satisfied (after renaming $t_1, \ldots, t_n$) THEN
                      evaluate the SELECT clause and generate the result tuple (projection).

Comparison

- The relational algebra has no notion of grouping and aggregate functions
- SQL has no clause that corresponds to relational division

Example 4.17
Consider again Example 4.10 (Slide 86).

The corresponding SQL formulation that implements division corresponds to the textual

"all countries that occur in $\pi[country](enc)$, with the additional condition that they occur in enc together with each of the $continent$ values that occur in cts",

or equivalent

"all countries c in $\pi[country](enc)$ such that there is no $continent$ value $cont$ in cts such that c does not occur together with $cont$ in enc": □

Example 4.17 (Continued)

"all countries c *in* $\pi[country](enc)$ *such that there is no* $continent$ *value* $cont$ *in* cts *such that* c *does not occur together with* $cont$ *in* enc*":*

SELECT enc1.country
FROM enc AS enc1 — *consider enc1.country="R" and enc1.country="D"*
WHERE NOT EXISTS — *correlated subquery*

((SELECT ct — *always*
FROM cts)

"Europe"
"Asia"

MINUS
(SELECT ct
FROM enc AS enc2
WHERE enc1.country = enc2.country
)

for "R":

"R"	"Asia"
"R"	"Europe"

for "D":

"D"	"Europe"

) — *remains: for "R": nothing* $\leadsto$ *"R" belongs to the result*
for D: "Asia" $\leadsto$ *"D" does not belong to the result*

Orthogonality

Full orthogonality means that an expression that results in a relation is allowed everywhere, where an input relation is allowed

- subqueries in the FROM clause
- subqueries in the WHERE clause
- combinations of set operations

But:

- Syntax of aggregation functions is not fully orthogonal:
  Not allowed: SUM(SELECT ...)

```
SELECT SUM(pop_biggest)
   FROM (SELECT country, MAX(population) AS pop_biggest
            FROM City
            GROUP BY country);
```

- The language OQL (Object Query Language) uses similar constructs and is fully orthogonal [see Lecture *Information Systems*].

Chapter 5
Efficiency and Optimization

Physical Data Organization

- the **conceptual schema** defines which data is described and its semantics.
- the **physical schema** defines the **physical database** where the data is actually stored.
  $\Rightarrow$ efficiency
- system: the data is actually stored in **files**: data that semantically belongs together (a relation, a part of a relation (**hashing**), some relations (**cluster**)).
- additionally, there are files that contain auxiliary information (**indexes**).
- data is accessed **pagewise** or **blockwise** (typically, 4KB – 8KB).
- each page contains some **records** (tuples). Records consist of **fields** that are of an elementary type, e.g., bit, integer, real, string, or pointer.

Database Access Mechanism

Records must be loaded from (and written to) the secondary memory for processing:

- the **file manager** determines the page where the record is stored.
- the **buffer/cache manager** is responsible to provide the page in the buffer (**buffer management**):
  - maintains a **pool** of pages (organized as frames).
    for every page, it is stored if the page has been changed, how often/frequently it has been used, and if it is currently used by transactions
  - if the required page is not in the cache, some stored page is replaced (if it has been changed, it must be written to the secondary memory)
- complex **prefetching** strategies, based on knowledge about transactions.
  [see lecture on Operating Systems]

Storage of Files, Pages, and Records

- Inside a file, every tuple/record has a **tuple identifier** of the form (p, n) where p is the page number and n is its index inside the page.
  Each page then contains an **directory** that assigns a physical address to each n.
- memory management for deleted records
- different strategies for fixed-length and variable-length records

... so far to the physical facts ...

5.1 Efficient Data Access

- efficiency depends on the detailed organization and additional algorithms and data structures
- support **generic** operations:
  - **Scan:** all pages that contain records are read.
  - **Equality Search:** all records that satisfy some equality predicate are read.
    SELECT * FROM City WHERE Country = “D”;
  - **Range Search:** all records that satisfy some comparison predicate are read.
    SELECT * FROM City WHERE Population > 100.000;
  - **Modify**, **Delete**: analogously
  - **Insert**: analogously: search for an appropriate place where to put the record.
- linear search (scan) ??
- Need for efficient searching (equality and/or range)

INDEXING

Indexes (for a file) are auxiliary structures that support special (non-linear) **access paths**

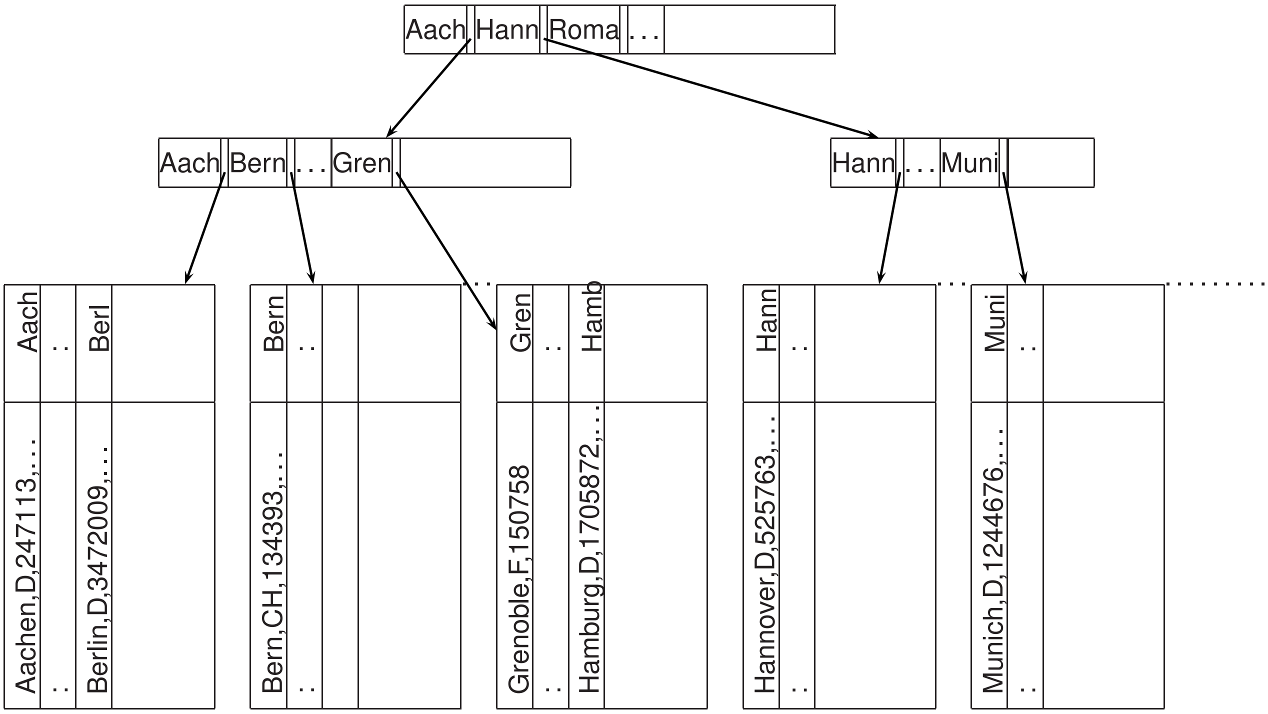
- Based on **search keys**
- not necessarily the relational "keys", but *any* combination of attributes
- a relation may have several search keys
- an *index* is a set of data entries on some pages together with efficient access mechanisms for locating an entry according to its search key value.
- different types of indexes, depending on the operations to be supported:
  - equality search
  - range search (ordered values)
  - search on small domains

B-TREES

A **B-tree** of order (m, l), $m \geq 3$, $l \geq 1$, is a search tree [see lecture on Algorithms and Data Structures]:

- the root is either a leaf or it has at least 2 children
- every inner node has at least $\lceil m/2 \rceil$ and at most m children
- all leaves are on the same level (balanced tree)
- inner nodes have the form $(p_0, k_1, p_1, k_2, p_2, \ldots, k_n, p_n)$ where $\lceil m/2 \rceil - 1 \leq n \leq m - 1$ and
  - k_i are search key values, ordered by $k_i < k_j$ for $i < j$
  - p_i points to the $i + 1$th child
  - all search key values in the left (i.e., p_{i-1}) subtree are less than the value of k_i (and all values in the right subtree are greater or equal)
- The leaves are of the form $([k_1, s_1], [k_2, s_2], \ldots, [k_g, s_g])$ where $g \leq l$, k_i is a search key value, and s_i is the corresponding record.

Example B-Tree



Properties

- Let N the number of entries. Then, for the height h of the tree, $h \leq \lceil log_{m/2}(2N/l) \rceil$ (l/2 entries per leaf, inner nodes half filled) and $h \geq \lceil log_m(N/l) \rceil$ (l entries/leaf, inner nodes completely filled) (equality search needs h steps)
- if the leaves are connected by pointers, ordered sequential access (range search) is also supported
- insertions and modifications may be expensive (tree reorganization)

Example

Example 5.1
Consider the MONDIAL *database with 3000 cities, with an index over the name. Assume the following sizes:*

- *every leaf (tuple) page contains 100 cities,*
- *every inner node contains 10 pointers*

Then

- *every inner node on the lowest level covers 1000 cities*
- *only one level of inner nodes*
- *access every city with WHERE Name ="..." in 3 steps*
- *index on population, e.g., for WHERE Population > 1.000.000 ORDER BY Population* □

HASH-INDEX (DICTIONARY)

The file is distributed over k **tiles**, depending on the search key values.

- each tile consists of one or more pages.
- A **hash function** h is a function that maps each search key value to a tile number. operation: lookup(keyvalue)
- common technique: convert search key value to an integer i. i mod k gives the tile number.

Example 5.2
Consider a hash index on City.(name,province,country). h computes the sum of the ASCII numbers of the letters and takes the remainder modulo 111. □

Properties:

- equality search and insert in constant time (+ time for searching in the tile)
- does not support range queries or ordered output

Bit Lists

If the number of possible values of a search key value is small wrt. the number of tuples, **bit lists** are useful as indexes

Let a an attribute of the records stored in a file f, where k values of f exist. Then, a bit list index to f consists of k bit vectors $B(v_i)$, $i \leq 1 \leq k$.

- $B(v_i)(j) = 1$ if the jth record in f has the value v_i for the attribute a.

Properties:

- access all tuples with a given value:
  without bit list: linear search over all pages
  with bit list: access bit list, and access those pages where a 1-tuple is located.
- modifications: no problem
- deletions: depends (if gaps are allowed on the pages)

Example 5.3
Consider the relation is_member(organization, country, type) where type has only the values "member", "applicant", and "observer":

is_member		
Org.	***Country***	***Type***
EU	*D*	*member*
UN	*D*	*member*
UN	*CH*	*observer*
UN	*R*	*member*
EU	*PL*	*applicant*
EU	*CZ*	*applicant*
:	*:*	*:*

Bit list for type=member: 1 1 0 1 0 0 ...
Bit list for type=observer: 0 0 1 0 0 0 ...
Bit list for type=applicant: 0 0 0 0 1 1 ...

- *Search for members of the UN:*
  without bit list: linear search over all pages
  with bit list: access bit list, and access those pages where a 1-tuple is located. □

CLUSTERING

Data from the secondary storage is fetched by **pages**.
Pages are cached in the main memory.

- keep data that semantically belongs together on the same pages
- obviously done for relations
- data of several relations $R_1(\bar{X}_1), \ldots, R_n(\bar{X}_n)$ can also be grouped to **clusters**:
  - choose a set $\bar{Y} \subseteq \bar{X}_1 \cap \ldots \cap \bar{X}_n$ as **cluster key**.
  - combine relations by their $\bar{Y}$-values.

Example 5.4
Consider the relation schemata
organization(name,abbrev,established,...)
and
is_member(organization, country, type).
The foreign key
is_member.organization → organization.abbrev
is used as cluster key.

Organization		
Abbrev		
EU	***Name***	***established***
	European Union	07.02.1992
	Country	***Type***
	D	member
	A	member
	:	:
UN	***Name***	***established***
	United Nations	26.06.1945
	Country	***Type***
	D	member
	CH	observer
	:	:

5.2 Efficient Query Evaluation

Queries are formulated *declaratively* (e.g., SQL or algebra trees), actually built over a small set of basic operations (cf. the definition of the relational algebra).

Semantical optimization: consider integrity constraints in the database.

Example: $population > 0$, thus, a query that asks for negative values can be answered without explicit computation.

- not always obvious
- general case: first-order theorem proving.
- special cases: [see lecture on Database Theory]

Logical/algebraic optimization: search for an equivalent algebra expression that performs better:

- size of intermediate results,
- implementation of operators as algorithms,
- presence of indexes and order.

Basic Techniques

Iteration: all tuples in the input relations are processed iteratively. If possible, instead of the tuples themselves, an index can be used.

Indexes: selections and joins can be based on processing of an index for determining the tuples that satisfy the selection condition (a join condition is also a selection condition).

- a **hash index** to a conjunction of **equality** predicates of the form **field = value** can be used if every field of the search key occurs exactly in one predicate together with a constant,
- a **tree index** to a conjunction of **comparison** predicates of the form **field** θ **value** if a prefix of the search key exists such that each field of the prefix occurs in exactly one predicate together with a constant.
  Example: if (A_1, A_2, A_3) is the search key, (A_1, A_2) is a prefix.

5.2.1 Algebra Operations

SELECTION

Selection: $\sigma[R.field\,\theta\,value]R$.

- no index, unordered tuples: linear scan of the file
- no index, tuples ordered wrt. $field$: find the tuple(s) that satisfy "$field = value$" (binary search) and process the remaining tuples.
- tree index: find the tuple(s) that satisfy "$field = value$" using the tree index and process the remaining tuples.
- hash: suitable only if θ is equality.

+ strategies for compound selection conditions.

PROJECTION

Projection: $\pi[field_1, field_2, ..., field_m]R$.

Main problem: remove duplicates (relational algebra does not allow duplicates, SQL does).

- by sorting:
  scan the file, create a new file with projected tuples.
  sort the new file (over all fields, $n \log n$).
  scan the result, remove duplicates.
- by hash: scan the file, put the projected tuples into a hash table.
  duplicates all end up in the same tile (still in files).
  remove duplicates separately for each tile (iterating the hashing process with different functions until tile fits in main memory)
  collect the tiles in the output file.

JOIN

Consider an Equijoin: $R \bowtie_{R.A=S.B} S$.

Nested-loop-join

$\texttt{foreach}$ tuple $r \in R$ $\texttt{do}$

  $\texttt{foreach}$ tuple $s \in S$ $\texttt{do}$

    $\texttt{if}$ $r_A = s_B$ $\texttt{then}$ add $[r, s]$ to result

- very inefficient. Can be "optimized" by page-oriented strategies:

  block-nested-loop-join: process the smaller relation in inner loop to keep it in main memory. If not possible, divide it into blocks that fit in main memory and process each of the blocks separately:
  When reading a block, generate a temporary index over the join attribute(s) that is used in the comparison.

- **index-nested-loop-join:** if for one of the input relations there is an index over the join attribute, choose it as the inner relation.

- **parallelization:** divide one of the relations into partitions and process each of them on a separate processor.

Sort-merge-join

Algorithm type: "Scan line"

- sort both relations according to the join attributes,
- search for matches as follows:
  - proceed through the ordered relations R and S stepwise, always doing a step in the relation where the "smaller" value is.
- if a match is found:
  - generate a result tuple.
  - check for tuples which have the same values of the join attributes (must immediately follow this match in both relations).

Sort-merge-join: Algorithm

```
if R not sorted on attribute A, sort it;
if S not sorted on attribute B, sort it;
Tr := first tuple in R;
Ts := first tuple in S;
Gs := first tuple in S;
while Tr ≠ eof and Ts ≠ eof do {
    while Tr ≠ eof and Tr.A < Ts.B do   Tr = next tuple in R after Tr;
    while Ts ≠ eof and Tr.A > Ts.B do   Ts = next tuple in S after Ts;
    // now, Tr.A = Ts.B: match found
    while Tr ≠ eof and Tr.A = Ts.B do {
        Gs := Ts;
        while Gs ≠ eof and Gs.B = Tr.A do {
            add [Tr, Gs] to result;
            Gs = next tuple in S after Gs;}
        Tr = next tuple in R after Tr;
        }
    Ts := Gs;
}
```


Hash-join

Algorithm type: "Divide and Conquer"

Partitioning (building) phase:

- Partition the smaller relation R by a hash function h_1 applied to $R.A$.
- Partition the larger relation S by the same hash function applied to $S.B$.
  (into different hash tables)

Matching (probing) phase:

- potential(!) matches have been mapped to "face-to-face" partitions.
- thus, consider each pair of corresponding partitions:
- if partition of R does not fit into main memory, proceed recursively with another h.
- otherwise – i.e., R-partition fits in main memory:
  - if corresponding S-partition also fits into main memory, compute the join.
  - otherwise proceed recursively with another (finer) h_2 inside main memory and process S-partition pagewise.

Exercise

Exercise 5.1

Consider the join of two relations R and S wrt. the coin condition $R_i = S_j$. R uses M pages with p_R tuples each, and S uses N pages with p_S tuples each. Let $M = 1000$, $p_R = 100$, $N = 500$, $p_S = 80$; the cache can keep 100 pages. Compute the number of required I/O-operations for computing the join (without writing the result relation) for:

(a) simple nested-loop-join,

(b) pagewise nested-loop-join,

(c) index-nested-loop-join (Index on S_j),

(d) sort-merge-join,

(e) hash-join. □

Set Operations: union, difference, intersection, cartesian product

- intersection and cartesian product are special cases of join
- union and difference: analogous to sort-merge-join or hash-join

Aggregation: SUM, AVG, COUNT, MIN, MAX

- SUM, AVG, COUNT: linear scan necessary (need to consider all tuples)
- MIN, MAX: index can be used
- support for grouping (SQL: GROUP BY) using a hash table

5.2.2 Algebraic Optimization

The operator tree of an algebra expression provides a base for several optimization strategies:

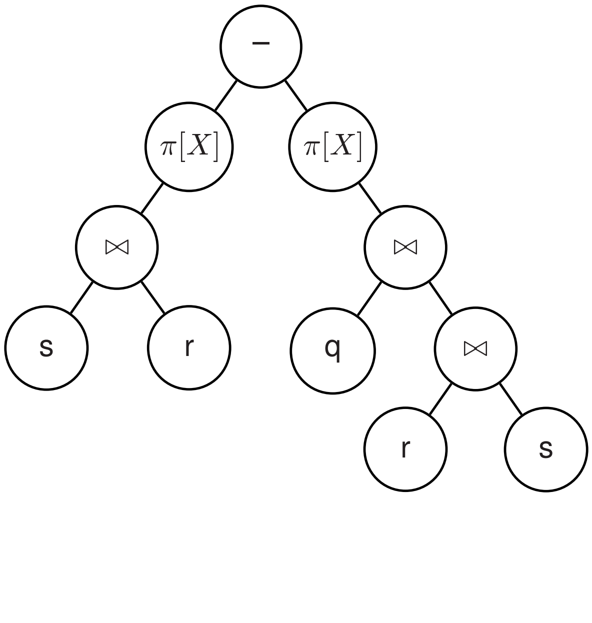
- reusing intermediate results
- equivalent restructuring of the operator tree
- "shortcuts" by melting several operators into one (e.g., join + equality predicate $\rightarrow$ equijoin)
- combination with actual situation: indexes, properties of data

Real-life databases implement this functionality.

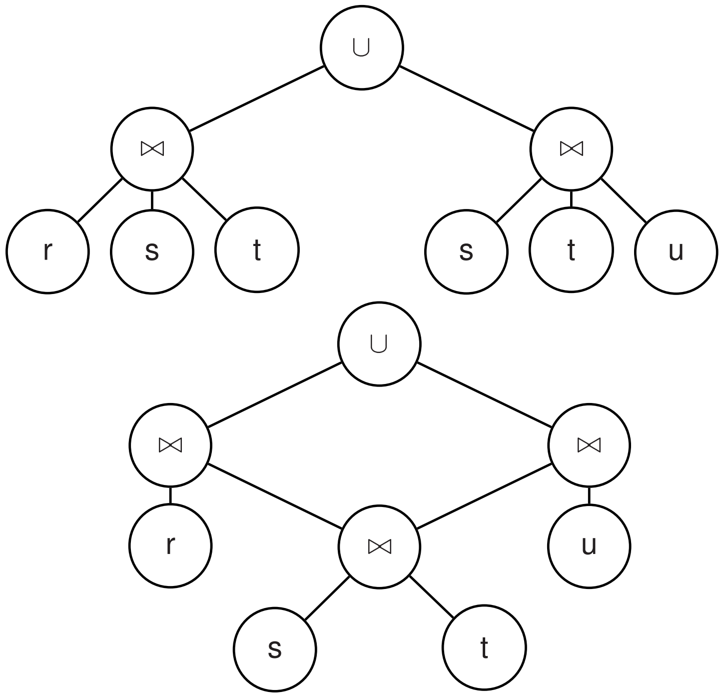
- SQL: **declarative** specification of a query
- internal: algebra tree + optimizations

Reusing intermediate results

- Multiply occurring subtrees can be reused (directed acyclic graph (DAG) instead of algebra tree)



Reusing intermediate results



OPTIMIZATION BY TREE RESTRUCTURING

- Equivalent transformation of the operator tree that represents an expression
- Based on the equivalences shown on Slide 101.
- minimize the size of intermediate results
  (reject tuples/columns as early as possible during the computation)
- selections reduce the number of tuples
- projections reduce the size of tuples
- apply both as early as possible (i.e., before joins)
- different application order of joins

Push Selections Down

Assume $r, s \in \text{Rel}(\bar{X})$, $\bar{Y} \subseteq \bar{X}$.

$$\sigma[cond](\pi[\bar{Y}](r)) \equiv \pi[\bar{Y}](\sigma[cond](r))$$

(condition: $cond$ does not use attributes from $\bar{X} - \bar{Y}$, otherwise left term is undefined)

$$\sigma_{pop>1E6}(\pi[name, pop](country)) \equiv \pi[name, pop](\sigma_{pop>1E6}(country))$$

$$\sigma[cond](r \cup s) \equiv \sigma[cond](r) \cup \sigma[cond](s)$$

$$\sigma_{pop>1E6}(\pi[name, pop](country) \cup \pi[name, pop](city)) \equiv \sigma_{pop>1E6}(\pi[name, pop](country)) \cup \sigma_{pop>1E6}(\pi[name, pop](city))$$

$$\sigma[cond](\rho[N](r)) \equiv \rho[N](\sigma[cond'](r))$$

(where $cond'$ is obtained from $cond$ by renaming according to N)

$$\sigma[cond](r \cap s) \equiv \sigma[cond](r) \cap \sigma[cond](s)$$

$$\sigma[cond](r - s) \equiv \sigma[cond](r) - \sigma[cond](s)$$

π : see comment above. Optimization uses only left-to-right.

Push Selections Down (Cont'd)

Assume $r \in \text{Rel}(\bar{X})$, $s \in \text{Rel}(\bar{Y})$. Consider $\sigma[cond](r \bowtie s)$.

Let $cond = cond_{\bar{X}} \wedge cond_{\bar{Y}} \wedge cond_{\overline{XY}}$ such that

- $cond_{\bar{X}}$ is concerned only with attributes in $\bar{X}$
- $cond_{\bar{Y}}$ is concerned only with attributes in $\bar{Y}$
- $cond_{\overline{XY}}$ is concerned both with attributes in $\bar{X}$ and in $\bar{Y}$.

Then,

$$\sigma[cond](r \bowtie s) \equiv \sigma[cond_{\overline{XY}}](\sigma[cond_{\bar{X}}](r) \bowtie \sigma[cond_{\bar{Y}}](s))$$

Example 5.5
Names of all countries that have been founded earlier than 1970, their capital has more than 1.000.000 inhabitants, and more than half of the inhabitants live in the capital. □

Example 5.5 (Continued)
(Solution)

$$\begin{aligned}
&\pi[Name](\sigma[establ < \text{“01 01 1970”} \wedge city.pop > 1.000.000 \wedge country.pop < 2 \cdot city.pop]\\
&\quad (country \times_{country.(capital,prov,code)=city(name,prov,country)} city)\\
&\equiv \pi[Name](\sigma[country.pop < 2 \cdot city.pop]\\
&\qquad (\sigma[establ < \text{“01 01 1970”}](country)\\
&\qquad\qquad \times_{country.(capital,prov,code)=city(name,prov,country)}\\
&\qquad\qquad\qquad \sigma[city.pop > 1.000.000](city)))
\end{aligned}$$

□

- Nevertheless, if $cond$ is e.g. a complex mathematical calculation, it can be cheaper first to reduce the number of tuples by $\cap$, $-$, or $\bowtie$

$\Rightarrow$ data-dependent strategies (see later)

Push Projections Down

Assume $r, s \in$ Rel$(\bar{X})$, $\bar{Y} \subseteq \bar{X}$.

Let $cond = cond_{\bar{X}} \wedge cond_{\bar{Y}}$ such that

- $cond_{\bar{Y}}$ is concerned only with attributes in $\bar{Y}$
- $cond_{\bar{X}}$ is the remaining part of $cond$ that is also concerned with attributes $\bar{X} \setminus Y$.

$$\begin{aligned}
\pi[\bar{Y}](\sigma[cond](r)) &\equiv \sigma[cond_{\bar{Y}}](\pi[\bar{Y}](\sigma[cond_{\bar{X}}](r)))\\
\pi[\bar{Y}](\rho[N](r)) &\equiv \rho[N](\pi[\bar{Y}'](r))\\
&\text{(where } \bar{Y}' \text{ is obtained from } \bar{Y} \text{ by renaming according to } N\text{)}\\
\pi[\bar{Y}](r \cup s) &\equiv \pi[\bar{Y}](r) \cup \pi[\bar{Y}](s)
\end{aligned}$$

- Note that this does *not* hold for “$\cap$” and “$-$”!
- advantages of pushing “σ” vs. “π” are data-dependent
  Default: push σ lower.

Assume $r \in$ Rel$(\bar{X})$, $s \in$ Rel$(\bar{Y})$.

$$\pi[\bar{Z}](r \bowtie s) \equiv \pi[Z](\pi[\bar{X} \cap \overline{ZY}](r) \bowtie \pi[\bar{Y} \cap \overline{ZX}](s))$$

- complex interactions between reusing subexpressions and pushing selection/projection

Operator Evaluation by Pipelining

- above, each algebra operator has been considered separately
- if a query consists of several operators, the materialization of intermediate results should be avoided
- **Pipelining** denotes the immediate propagation of tuples to subsequent operators

Example 5.6

- *$\sigma[A = 5 \wedge B > 6]R$:*

  Assume an index that supports the condition $A = 5$.

  - *without pipelining: compute $\sigma[A = 5]R$ using the index, obtain R'. Then, compute $\sigma[B > 6]R'$.*
  - *with pipelining: compute $\sigma[A = 5]R$ using the index, and check **on-the fly** each qualifying tuple against $\sigma[B > 6]$.* □

- **Unary** (i.e., selection and projection) operations can always be pipelined with the next lower binary operation (e.g., join)

Example 5.6 (Continued)

- *$\sigma[cond](R \bowtie S)$:*
  - *without pipelining: compute $R \bowtie S$, obtain RS, then compute $\sigma[cond](RS)$.*
  - *with pipelining: during computing $(R \bowtie S)$, each tuple is immediately checked whether it satisfies $cond$.*
- *$(R \bowtie S) \bowtie T$:*
  - *without pipelining: compute $R \bowtie S$, obtain RS, then compute $RS \bowtie T$.*
  - *with pipelining: during computing $(R \bowtie S)$, each tuple is immediately propagated to one of the described join algorithms for computing $RS \bowtie T$.* □

Most database systems use iterator-based implementation of algebra operators with pipelining.

SITUATION-DEPENDENT OPTIMIZATION

Determining the optimal **execution plan** depends on **cost models**, **heuristics**, and the actual physical schema and the actual database contents:

- index structures
- statistics on data (i.e., to some extent state dependent)
  - cardinality of relations
  - distribution of values

Notions

The **selectivity** sel of operations can be used for estimating the size of the result relation:

- Selection with condition p:

$$sel_p = \frac{|\sigma[p](R)|}{|R|}$$

  Proportion of tuples that satisfy the selection condition.
  - for p an equality test $R.A = c$ where A is a key of R, $sel_p = 1/|R|$.
  - if $R.A$ distributes evenly on i different values, $sel_p = 1/i$.

- Join of R und S:

$$sel_{RS} = \frac{|R \bowtie S|}{|R \times S|} = \frac{|R \bowtie S|}{|R| \cdot |S|}$$

  is the propertion of result tuples wrt. the cartesian product.

  for $R \bowtie_{R.A=S.B} S$ with A a key attribute, $|R \bowtie_{R.A=S.B} S| \leq |S|$, thus $sel_{RS} \leq 1/|R|$.

- optimal order of applying joins
- optimal order of applying selections

Application-Level Algorithmic Optimization

And never forget about using efficient algorithms for querying the database!

- analyze the problem from the algorithmic point of view
- before hacking

Examples

Use the MONDIAL database for the following examples.

Example 5.7
Compute all pairs of european countries that are adjacent to the same set of seas. □

Example 5.8
Compute all political organizations that have at least one member country on every continent (this operation is called relational division*).* □

5.2.3 Exercises and Examples

Solution to Exercise 5.1

R: 1000 pages, each of them with 100 tuples
S: 500 pages, each of them with 80 tuples
Cache: 100 pages

a) simple nested loop join
  - outer loop: R
    Load each page of R. For all tuples, iterate over S's pages.
    $1000 \cdot (1$ (load) $+ 100$ (tuples per page) $\cdot 500$ (pages of S)) $= 50,001,000$
  - outer loop: S
    Load each page of S. For all tuples, iterate over R's pages.
    $500 \cdot (1$ (load) $+ 80$ (tuples per page) $\cdot 1000$ (pages of R)) $= 40,000,500$

b) pagewise nested loop:
  - outer loop: R. Load each page of R. Combine all tuples of that page with all tuples from each page of S.
    $1000 \cdot (1 + 500$ (pages of S)) $= 501,000$
  - outer loop: S. $500 \cdot (1 + 1000$ (pages of R)) $= 505,000$

Solution to Exercise 5.1 (cont'd)

R: 1000 pages, each of them with 100 tuples
S: 500 pages, each of them with 80 tuples
Cache: 100 pages

c) Index-nested-loop (index on S_j): Assumptions:
   - every S_j occurs – average – $k = 4$ times.
   - index depth 3

   Iterate over R, for each tuple search for the S_j key and access the tuples
   $1000 \cdot (1 \text{ (load)} + 100 \cdot (3 \text{ (search)} + 4 \text{ (access tuples)})) = 701,000$
   in case that R is ordered wrt. R_i , much better ...

d) sort-merge: sort R according to $R.i$:
   sort each 100 pages in place (Quicksort), store them: $10 \cdot 100 \cdot 2 = 2000$
   mergesort the sorted pages (linear) – read and write: 2000
   sort S according to S_j analogously: 2000
   merge-join of sorted relations: linear scan: $1000 + 500 = 1500$
   Overall: 7500

Solution to Exercise 5.1 (cont'd)

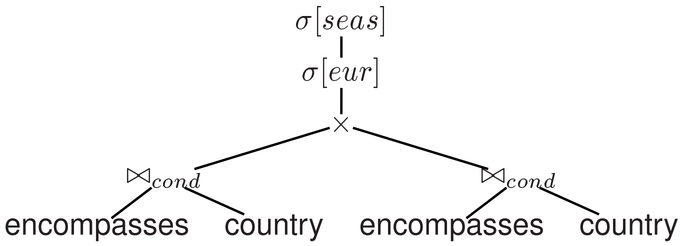
R: 1000 pages, each of them with 100 tuples
S: 500 pages, each of them with 80 tuples
Cache: 100 pages

e) Hash-Join:
   - Hash R: read 33 pages-wise into 66 tiles. Do this 30 times, collect in 66 partitions.
     In the average, each partition them contains about 15 pages.
     (about $30 \cdot (33 + 66) = 3000$ accesses).
   - Same for S. Get 66 partitions of average 7.5 pages. (about 1000 accesses).
   - now we have 66 corresponding pairs of partitions (one from R, one from S.
     Join each pair: read $15 + 8$ (overestimate) pages per partition and process them.
     (about $66 \cdot (15 + 8) = 1500$ accesses).

   Overall: 5500 accesses.

Solution to Exercise 5.7

First, compute all pairs of european countries. Note that we are interested in their names, thus we need also the *Country* relation.



- $cond =$ “enc.country=country.code”.
- $eur =$ “enc1.continent=‘europe’ and enc1.continent=‘europe’”.
- $seas$ expresses that both countries border the same set of seas.
  it is a correlated subquery.
- add suitable projections.
- move $\sigma[eur]$ downwards both sides directly to *encompasses*
- obviously, both subtrees of $\times$ are identical.

Solution to Exercise 5.7 (cont’d)

- $\sigma[seas(C_1, C_2)]$ is a correlated subquery that takes two country codes as input:

$$\begin{aligned}\sigma[seas(C_1, C_2)] &= seas(C_1) - seas(C_2) = \emptyset \wedge seas(C_2) - seas(C_1) = \emptyset \\ &= (seas(C_1) - seas(C_2)) \cup (seas(C_2) - seas(C_1)) = \emptyset\end{aligned}$$

- $seas(C) = \pi[sea](\sigma[country = C](geo\_sea))$
- for each country, $seas(C)$ is computed only once an then reused.

Resulting SQL skeleton (using subqueries in the FROM clause):

```
SELECT ...
FROM (SELECT european countries) as C1,
     (SELECT european countries) as C2
WHERE σ[seas(C1, C2)]
```

Solution to Exercise 5.8

- $\pi[abbrev](\sigma[all\_continents(org)])(organization)$ where $\sigma[all\_continents(org)]$ is true for

$$\{org \mid \forall cont : cont \text{ is a continent} \rightarrow org \text{ has a member on } cont\}$$

- convert $\forall$ into $\neg exists$:

$$\{org \mid \neg\exists cont : cont \text{ is a continent and } org \text{ has no member on } cont\}$$

- Thus, $\sigma[all\_continents(org)]$ checks if

  $\pi[name](continent) - \pi[enc.continent]((\sigma[organization = org]is\_member) \bowtie encompassed)$

  is empty.

Resulting SQL skeleton (uses a correlated subquery):

```
SELECT ...
FROM organization
WHERE NOT EXISTS
      ((SELECT continents)
       MINUS
       (SELECT continents where org has a member))
```


Chapter 6
Running a Database: Safety and Correctness Issues

- Transactions
- Safety against failure

Not discussed here:

- Access control, Authentication

6.1 Transactions: Properties and Basic Notions

Transaction:

- a unit of work from the user's point of view.
- for the DBS: a process, characterized by a sequence of database accesses
- requirements: **ACID-properties**:

Atomicity: A transaction is (logically) a unit that cannot be further decomposed: its effect is *atomic*, i.e., all updates are executed completely, or nothing at all ("all-or-nothing").

Concistency: A transaction is a correct transition from one state to another. The final state is not allowed to violate any integrity condition (otherwise the transaction is undone and rejected).

Isolation: Databases are multi-user systems. Although transactions are running *concurrently*, this is hidden against the user
(i.e., after starting a transaction, the user does not see changes by other transactions until finishing his transaction, *simulated* single-user).

Durability: If a transaction completes successfully, all its effects are durable (=persistent). I.e., no error situation (including system crash!) is allowed to undo them $\Rightarrow$ safety.

Transactions consist of elementary actions:

- Read access: READ
  By READ A (RA), the value of a DB-object A from the DB is copied to the local workspace of the transaction.
- Write Access: WRITE
  By WRITE A (WA), the value of a DB-object A is copied from the local workspace of the transaction to the DB.
- BEGIN WORK and COMMIT WORK denote its begin (BOT - begin of transaction) and its successful completion (EOT - end of transaction).

$\Rightarrow$ of the form BOT RA RB RC ... WA ... RD ... WE EOT

- ROLLBACK WORK for undoing all its effects (ABORT).
- These elementary actions are *physically* atomic. At every timepoint, only one such action is executed.
- in contrast, several transactions may be **interleaved** (see below).

6.2 Transaction models

FLAT TRANSACTIONS

Basic transaction model: Transactions are a “flat” (and short) sequence of elementary actions without additional structure.

Example 6.1

Outline of a simple transaction for transferring money from account A to account B:

1. *BEGIN WORK*
3. *debit (READ and WRITE) money from account A.*
4. *book money (READ and WRITE) on account B.*
5. *if account A negative, then ROLLBACK, otherwise COMMIT WORK.* □

FLAT TRANSACTIONS WITH SAVEPOINTS

Limits of simple flat transactions: long transactions, e.g., travel booking (hotel, several flights, rental car)

- partial rollback, for trying alternative continuations:
- SAVE WORK defines savepoints (intermediate states)
- sequences between savepoints are *atomic* (but in general not consistent and durable)
- ROLLBACK WORK(i) undoes effects back to savepoint i
- COMMIT WORK commits the whole transaction (ACID)
- complete ROLLBACK WORK undoes the whole transaction

Other Transaction Models

- Nested transactions

6.3 Multi-User Aspects

- In general, at any timepoint, several transactions are running.
- means **interleaving** (i.e., one step here, one step there, and again one step here)
- not necessarily true **parallelism** (requires multi-processor systems)
- techniques for interleaving also sufficient for parallelism

Goal of multi-user policies: allow for as much interleaving as possible without the risk of "unintended" results

Problem: transactions run on **shared** data.
⇒ enforce virtual isolation

TYPICAL ERROR SITUATIONS

Lost update

population update	population update
SELECT population INTO pop	
FROM City WHERE name="Berlin"	
	SELECT population INTO pop
	FROM City WHERE name=Berlin
pop := pop + 2000;	
	pop := pop + 1000;
UPDATE City	
SET population = pop	
WHERE Name = "Berlin"	
	UPDATE City
	SET population = pop
	WHERE Name = "Berlin"

- the first update is lost

Dirty Read

population update	population update
UPDATE City	
SET population = population + 1000	
WHERE Name = "Berlin"	
	SELECT population INTO pop
	FROM City
	WHERE Name = "Berlin"
⋮	pop := pop * 1.05;
	UPDATE City
	SET population = pop
	WHERE Name = "Berlin"
ABORT (i.e., ROLLBACK)	
(what must be done now?)	

- The second transaction reads and uses a value that has been undone

Non-repeatable Read

Sum of Population	Change Population
SELECT population INTO pop	
FROM City	
WHERE Name = "Berlin"	
sum := sum + pop	
	UPDATE City
	SET population= population + 1000
	WHERE Name = "Berlin"
	UPDATE City
	SET population= population + 2000
	WHERE Name = "Munich"
SELECT population INTO pop	
FROM City	
WHERE Name = "Munich"	
sum := sum + population	

- The value computed in sum does not correspond to any database state

Phantom

- sum of population of provinces equals population of country

Check sum of population	Insert new province
SELECT SUM(population) INTO sum FROM Province WHERE Country = “D”	/* people say that in the summer, the holiday island Mallorca (Spain) becomes a german province */
	INSERT INTO Province (name, country, population) VALUES (“Mallorca”, “D”, 100000)
	UPDATE Country SET population = population + 100000 WHERE Code = “D”
SELECT population INTO countrypop FROM Country WHERE Code = “D”	
IF countrypop $\neq$ sum THEN <error handling>	

- similar to non-repeatable read

6.4 Serializability

- A **schedule** wrt. a set of transactions is an interleaving of their (elementary) actions that does not change the inner order of each of the transactions.
- A schedule is **serial** if the actions of each individual transaction are immediately following each other (no interleaving).

Example 6.2 (Bank Accounts: Transferring Money from A to B)
T_1 = RA; A:=A-10; WA; RB; B:=B+10; WB, *T_2 = RA; A:=A-20; WA; RB; B:=B+20; WB*

Some schedules (without computation steps):

$$\begin{array}{ll}
S_1 = R_1A\ \ W_1A\ \ R_1B\ \ W_1B\ \ R_2A\ \ W_2A\ \ R_2B\ \ W_2B & \textit{(serial)} \\
S_2 = R_1A\ \ R_2A\ \ W_1A\ \ W_2A\ \ R_1B\ \ R_2B\ \ W_1B\ \ W_2B & \\
S_3 = R_1A\ \ W_1A\ \ R_2A\ \ W_2A\ \ R_1B\ \ W_1B\ \ R_2B\ \ W_2B & \\
S_4 = R_1A\ \ W_1A\ \ R_2A\ \ W_2A\ \ R_2B\ \ W_2B\ \ R_1B\ \ W_1B & \\
S_5 = R_2A\ \ W_2A\ \ R_2B\ \ W_2B\ \ R_1A\ \ W_1A\ \ R_1B\ \ W_1B & \textit{(serial)}
\end{array}$$

□

... which of them are “good”?

Serializability Criterion for Parallel Transactions

- "Isolation" requirement
  Transaction may not see results from other (not yet committed) ones
- The serial ones are good.
- are there other "good" ones?

Definition 6.1
*A schedule is **serializable** if and only if there exists an equivalent serial schedule.* □

Example 6.2 (Cont'd: Bank Accounts Interleaved)
A=B=10; T_1: RA; A:=A-10; WA; RB; B:=B+10; WB T_2: RA; A:=A-20; WA; RB; B:=B+20; WB

T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
RA		RA		RA		RA	
A:=A-10			RA	A:=A-10		A:=A-10	
WA		A:=A-10		WA		WA	
RB			A:=A-20		RA		RA
B:=B+10		WA			A:=A-20		A:=A-20
WB			WA		WA		WA
	RA	RB		RB			RB
	A:=A-20		RB	B:=B+10			B:=B+20
	WA	B:=B+10		WB			WB
	RB		B:=B+20		RB	RB	
	B:=B+20	WB			B:=B+20	B:=B+10	
	WB		WB		WB	WB	
serial		not serializable		serializable		serializable?	
A=-20, B=40		A=-10, B=30		A=-20,B=40		A=-20,B=40	
A+B=20		A+B=20		A+B=20		A+B=20	

□

Problem: what means "equivalence" in this context?

- consider each step in each transaction?
  Then, (4) is not equivalent with (1):
  in (1) T_1 reads $B = 10$, in (4), T_1 reads $B = 30$
- consider the initial and final database state?
  Then, (4) and (1) would be equivalent.

Exercise 6.1
Consider again Example 6.2 for A=B=10;
T_1*: RA; A:=A/2; WA; RB; B:=B/2; WB* *(Transforming DM to Euro)* *and*
T_2*: RA; A:=A-10; WA; RB; B:=B+10; WB* *(Money Transfer).*
Consider also the schedule T_2T_1*.* □

6.4.1 Formalization of the Semantics of Transactions

- Theory & algorithms depend only on the READ and WRITE actions, not on the semantics of the computations in-between.

  Transactions T and schedules S are represented as a sequence of their READ- and WRITE-Actions (actions are assigned to transactions by indexing).

Let $\mathcal{D}$ the domain of the database objects.

Let WA a write action of a transaction T and $RB_1, \ldots, RB_k$ $k \geq 0$ the read actions that are executed by T before WA.

The value written to A by WA is denoted by

$$f_{T,A}(B_1, \ldots, B_k),$$

where

$$f_{T,A} : \mathcal{D}^k \to \mathcal{D}$$

is called the **Interpretation** of the write action WA in T

($f_{T,A}$ encodes the functional relationship (computation) between the read-values and the written value)

Example 6.2 (Cont'd)

Consider again the transactions $T_1 = RA\ WA\ RB\ WB$ *and* $T_2 = RA\ WA\ RB\ WB$

Let the initial state be given by values A_0, B_0.

Schedule T_1T_2 (serial)			Schedule T_2T_1 (serial)		
$T_1:$	RA	A_0	$T_2:$	RA	A_0
	WA	$f_{T_1,A}(A_0)$		WA	$f_{T_2,A}(A_0)$
	RB	B_0		RB	B_0
	WB	$f_{T_1,B}(A_0, B_0)$		WB	$f_{T_2,B}(A_0, B_0)$
$T_2:$	RA	$f_{T_1,A}(A_0)$	$T_1:$	RA	$f_{T_2,A}(A_0)$
	WA	$f_{T_2,A}(f_{T_1,A}(A_0))$		WA	$f_{T_1,A}(f_{T_2,A}(A_0))$
	RB	$f_{T_1,B}(A_0, B_0)$		RB	$f_{T_2,B}(A_0, B_0)$
	WB	$f_{T_2,B}(f_{T_1,A}(A_0), f_{T_1,B}(A_0, B_0))$		WB	$f_{T_1,B}(f_{T_2,A}(A_0), f_{T_2,B}(A_0, B_0))$

□

Definition 6.2

Two schedules S, S' *(of the same set of transactions) are equivalent, if for every initial state and every interpretation of the writing actions,*

(1) all transactions read the same values in S *and* S', *and*

(2) S *and* S' *generate the same final states* □

Exercise 6.2

Consider again Example 6.2.

Show by the detailed tables with $f(...)$ *that Schedule* S_2 *and Schedule* S_4 *are not serializable, but Schedule* S_3 *is serializable.*

Give at least one more serializable schedule. □

6.4.2 Theoretical Investigations

Consider a schedule S together with two additional distinguished transactions T_0, T_∞:
T_0 generates the initial state, and T_∞ reads the final state of S.

- T_0 is a transaction that executes a write action for every database object for which S executes a read or write action.
- T_∞ is a transaction that executes a read action for every database object for which S executes a read or write action.

The schedule $\hat{S} = T_0\ S\ T_\infty$ is the **augmented schedule** to S.

Assumption (without loss of generality):

- each transaction reads and writes an object at most once,
- if a transaction reads *and* writes an object, then reading happens before writing.

Corollary 6.1
Two schedules S, S' (of the same set of transactions) are equivalent if and only if for every interpretation of the write actions, all transactions read the same values for $\hat{S}$ and $\hat{S}'$. □

Dependency Graphs

Consider a schedule S. The **D-Graph** (*dependency graph*) of S is a directed graph $DG(S) = (V, E)$ where V is the set of actions in $\hat{S}$ and E is the set of edges given as follows $(i \neq j)$:

- if $\hat{S} = \ldots R_iB \ldots W_iA \ldots$, then $R_iB \rightarrow W_iA \in E$,
  (i.e., T_i reads B (and possible uses it) and then writes a value A)
- if $\hat{S} = \ldots W_iA \ldots R_jA \ldots$, then $W_iA \rightarrow R_jA \in E$,
  if there is no write action to A between W_iA and R_jA in $\hat{S}$.
  (i.e., T_j reads a value A that has been written by T_i)

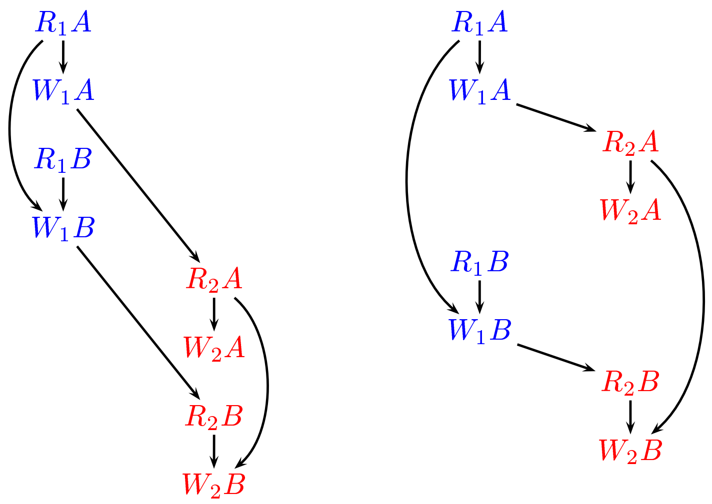
A transaction T' is **dependent** of a transaction T, if in S, either T' reads a value that has been written by T, or by a transaction that is dependent on T.

Example 6.3

Consider again Example 6.2: $T_1 = RA\ WA\ RB\ WB$; $T_2 = RA\ WA\ RB\ WB$

Consider the serial schedule T_1T_2 *and* $S_3 = R_1A\ W_1A\ R_2A\ W_2A\ R_1B\ W_1B\ R_2B\ W_2B$.

Dependency graphs:



□

Theorem 6.1

Two schedules S, S' (of the same transactions) are equivalent if and only if $DG(S) = DG(S')$. □

Conflict Graphs

Consider a schedule S. The **C-Graph** (*conflict graph*) of S is a directed graph $CG(S) = (V, E)$ where V is the set of Transactions in $\hat{S}$ and E is a set of edges give as follows ($i \neq j$):

- if $S = \ldots W_iA \ldots R_jA \ldots$ then $T_i \to T_j \in E$, if there is no write action to A between W_iA and R_jA in S **(WR-conflict)**.
- if $S = \ldots W_iA \ldots W_jA \ldots$ then $T_i \to T_j \in E$, if there is no write action to A between W_iA and W_jA in S **(WW-conflict)**.
- if $S = \ldots R_iA \ldots W_jA \ldots$ then $T_i \to T_j \in E$, if there is no write action to A between R_iA and W_jA in S **(RW-conflict)**.

Theorem 6.2

A schedule S is serializable, if $CG(S)$ is cycle-free. □

Proof Sketch: Since $CG(S)$ is cycle-free, there is a topological order of the nodes (i.e., of the transactions).
Let S' the serial schedule according to this ordering. Then, $DG(S) = DG(S')$.

Note that there are serializable schedules whose C-Graph contains cycles:

Example 6.4
Consider the following schedule S:

	T_1:			*RA*		*WB*	
S:	T_2:				*RB*		*WB*
	T_3:	*RC*	*WB*				

The C-Graph containing the edges (3,1), (3,2), (2,1), and (1,2) is cyclic.

Nevertheless, $S' = T_1 T_3 T_2$ *is an equivalent serial schedule.*

Why this? $W_1 B$ *is not used anywhere in* S*. It is also not used in* S'*.* □

Definition 6.3
A schedule is ***C-serializable*** *(conflict-serializable) if its C-Graph is cycle-free.* □

Note: In the sequel, serializability always means C-serializability.

6.4.3 Scheduler

The **Scheduler** of a database system ensures that only serializable schedules are executed. This can be done by different **strategies**.

Input: a set of actions of a set of transactions (to be executed)
Output: a serializable sequence (= the schedule to be actually executed) of these actions

Strategies

- **Based on the conflict graph:**
  Let S the current (partial) schedule and $action$ the next action of some transaction T.
  If $CG(S \cdot action)$ is cycle-free, then execute $action$. Otherwise abort T and all transactions that depend on T, and remove the corresponding actions from S.
- **Timestamps:**
  Each transaction T is associated to a unique timestamp $Z(T)$.
  (thus, transactions can be seen as ordered).
  Let S the current (partial) schedule and $action$ the next action of some transaction T.
  If for all transactions T' that have executed an action a' that is in conflict with $action$, $Z(T') \leq Z(T)$, then execute $action$. Otherwise abort T (T "comes too late") and all transactions that depend on T, and remove the corresponding actions from S.

Scheduling Strategies (Cont'd)

- **Optimistic Strategies:**
  (weaker than Timestamp – Assumption: "there is no conflict")

  Let S the current (partial) schedule. A transaction T is **active** in S, if an action of T is contained in S, and T is not yet completed.

  Let $readset(T), writeset(T)$ the set of objects that have been read/written by a transaction T.

  Let $action$ the next action, and T the corresponding transaction.

  Execute $action$ and update $readset(T), writeset(T)$.

  If $action$ is the final action in T, then check the following:

  – if for any other active transaction T':
    * $readset(T) \cap writeset(T') \neq \emptyset$,
    * $writeset(T) \cap writeset(T') \neq \emptyset$,
    * $writeset(T) \cap readset(T') \neq \emptyset$.

  then abort T and all transactions that depend on T, and remove the corresponding actions from S.

- **2-Phase Lock:** Can we enforce the above criterion?

6.5 Locks

- access to database objects is administered by **locks**
- Transactions need/hold locks on database objects:
  if T has a lock on A, T is has a privilege to use this object
- privileges allow for read-only, or read/write access to an object:
  – Read-privilege: RLOCK (L_RX)
  – Read and write-privilege: WLOCK (L_WX)
- operations:
  – LOCK X (LX): apply for a privilege for using X.
  – UNLOCK X (UX): release the privilege for using X.
- lock- and unlock operations are handled like actions and belong to the action sequence of a transaction.
- each action of a transaction must be inside a corresponding pair of lock-unlock-actions.
  (i.e., no action without having the privilege)

Example 6.5
Consider again Example 6.2: $T = RA\ WA\ RB\ WB$

Possible handling of locking actions:

- $T = LA\ RA\ WA\ UA\ LB\ RB\ WB\ UB$
- $T = LA\ LB\ RA\ WA\ RB\ WB\ UA\ UB$
- $T = LA\ RA\ WA\ LB\ RB\ WB\ UA\ UB$
- $T = LA\ RA\ WA\ LBUA\ RB\ WB\ UB$ □

Locking Policies

Locking policies (helping the scheduler) must guarantee correct execution of parallel transactions.

- privileges are given according to a **compatibility matrix**:
  Y: requested privilege can be granted
  N: requested privilege cannot be granted
- if there is only one privilege ("use an object"):

requested privilege — granted privilege (for the same object):

	LOCK
LOCK	N

- if read and write privilege are distinguished:

	RLOCK	WLOCK
RLOCK	Y	N
WLOCK	N	N

i.e., multiple transactions *reading* the same object are allowed.

PROBLEMS

- **Livelock**: It is possible that a transaction never obtains a requested lock (if always other transactions are preferred).
  Solution: e.g., first-come-first-served strategies
- **Deadlock**: during execution, deadlocks can occur:

  Transactions: T_1: LOCK A; LOCK B; RA WA RB WB UNLOCK A,B;
  T_2: LOCK B; LOCK A; RA WA RB WB UNLOCK A,B;

  Execution: T_1: LOCK A
  T_2: LOCK B

  Deadlock: no transaction can proceed.

Avoiding and resolving deadlocks

- each transaction applies for all required locks when starting (in an atomic action).
- a linear ordering of objects. Privileges must be requested according to this ordering.
- maintenance of a **waiting graph** between transactions: The waiting graph has an edge $T_i \to T_j$ if T_i applies for a privilege that is hold/blocked by T_j.
  - a deadlock occurs exactly if the waiting graph is cyclic
  - it an be resolved if one of the transactions in the cycle is aborted.

Note: locks alone do not yet guarantee serializability.

Example 6.6
Consider again Example 6.2 where T_1 and T_2 are extended with locks:

$T = LA\ RA\ WA\ UA\ LB\ RB\ WB\ UB$

Consider Schedule S_4 (which was not serializable):

$$S_{4L} = L_1A\ R_1A\ W_1A\ U_1A\quad L_2A\ R_2A\ W_2A\ U_2A$$
$$L_2B\ R_2B\ W_2B\ U_2B\ L_1B\ R_1B\ W_1B\ U_1B$$

□

Only correct use and policies do.

We need a protocol/policy that – if satisfied – *guarantees* serializability.

2-Phase Locking Protocol (2PL)

"After the first UNLOCK, a transaction must not execute any LOCK."

i.e., each transaction has a **locking phase** and an **unlocking phase**.

Example 6.7
Consider again Example 6.5:

Which transactions satisfy 2PL?

- *T = LA RA WA UA LB RB WB UB* (no)
- *T = LA LB RA WA RB WB UA UB* (yes)
- *T = LA RA WA LB RB WB UA UB* (yes)
- *T = LA RA WA LB UA RB WB UB* (yes!)

□

The last LOCK-operation of a transaction T defines T's **locking point**.

Theorem 6.3
The 2-Phase-Locking Protocol guarantees serializability. □

Proof: Consider a schedule S of a set $\{T_1, T_2, \ldots\}$ of two-phase transactions.

Assume that S is not serializable, i.e., $CG(S)$ contains a cycle, w.l.o.g. $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_k \rightarrow T_1$. Then, there are objects $A_1, \ldots, A_k$ such that

$$\begin{array}{l} S = \ldots\ (W_1 A_1)\ \ U_1 A_1\ \ \ldots\ L_2 A_1\ \ (R_2 A_1)\ \ \ldots \\ S = \ldots\ (W_2 A_2)\ \ U_2 A_2\ \ \ldots\ L_3 A_2 (R_3 A_2)\ \ \ldots \\ \qquad\qquad\qquad\vdots \\ S = \ldots\ (W_{k-1} A_{k-1})\ \ U_{k-1} A_{k-1}\ \ \ldots\ L_k A_{k-1}\ \ (R_k A_{k-1})\ \ \ldots \\ S = \ldots (W_k A_k)\ \ U_k A_k \ldots\ L_1 A_k\ \ (R_1 A_k) \ldots \end{array}$$

Let l_i the locking point of T_i. Then, the above lines imply that l_1 is before l_2, that is before l_3 etc, and l_{k-1} before l_k, that is before l_1. Impossible.

Properties of 2PL

2-Phase locking is optimal in the following sense:

For every non 2-phase transaction T_1 there is a 2-phase transaction T_2 such that for T_1, T_2 there exists a non-serializable schedule.

Example 6.8
Consider the non-2PL transaction from Example 6.7 and a 2PL transaction

$$T_1 = L_1 A\ \ R_1 A\ \ W_1 A\ \ U_1 A\ \ L_1 B\ \ R_1 B\ \ W_1 B\ \ U_1 B$$
$$T_2 = L_2 A\ \ L_2 B\ \ R_2 A\ \ W_2 A\ \ R_2 B\ \ W_2 B\ \ U_2 A\ \ U_2 B$$

The following schedule S (= S_4 from Examples 6.2) which has been shown not to be serializable:

$$\begin{array}{ll} S = & L_1 A\ \ R_1 A\ \ W_1 A\ \ U_1 A\ \ \ L_2 A\ \ L_2 B\ \ R_2 A\ \ W_2 A \\ & \qquad R_2 B\ \ W_2 B\ \ U_2 A\ U_2 B\ \ L_1 B\ \ R_1 B\ \ W_1 B\ \ U_1 B \end{array}$$ □

Properties of 2PL (Cont'd)

"optimal" does *not* mean that every serializable schedule can also occur under 2-phase locking:

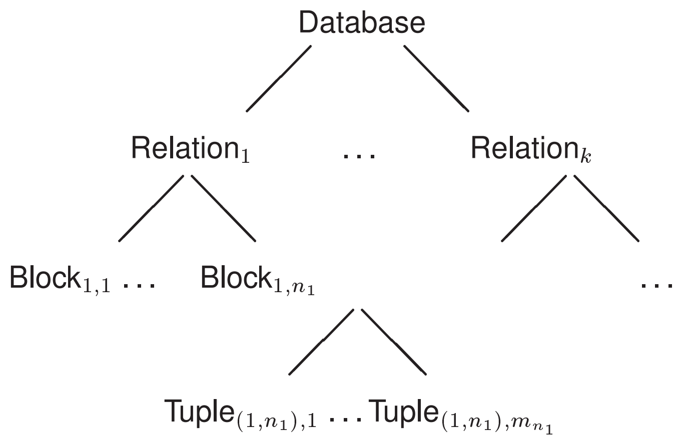
Example 6.9
The schedule S

$$S = R_1A\ R_2A\ W_2A\ R_3B\ W_3B\ W_1B$$

is serializable, but there is no way to add LOCK/UNLOCK actions to T_1, T_2 *that satisfy the 2PL requirement such that* S *is a schedule of these transactions.* □

Locking structured Data

- database consists of relations that are stored in blocks that contain tuples.



- find a compromise between maximal parallelism and number of locks.
- transactions that use all tuples of a relation: lock the relation
- transactions that lock only some tuples of a relation: lock the tuples.

LOCKING GRANULARITY

Having only tuple-locks and 2PL can still lead to non-serializable schedules:

Example 6.10
Consider again Slide 165.

T_1 *computes the sum of the population of all provinces of Germany – reading all tuples. Thus, at the beginning it locks all (existing) tuples.* T_2 *adds a new province and adapts the overall population.*

The schedule given on Slide 165 is still possible. □

Solution: Locking of complete tables, key areas, depending on predicates, or indexes.

Consequence: if the set of database objects changes dynamically, a conflict-based serializability test is not sufficient.

LOCKING IN THE SQL2-STANDARD

Serializability is enforced as follows:

- every transaction does only see updates by committed transactions
- no value that has been read/written by T can be changed by any other transaction before committing/aborting T.
  That means, "locks" are released only *after* commit (**strict 2-Phase Locking**).
- if T has read a set of tuples defined by some search criterion, this *set* cannot be changed until T is committed or aborted. (this excludes the phantom-problem)

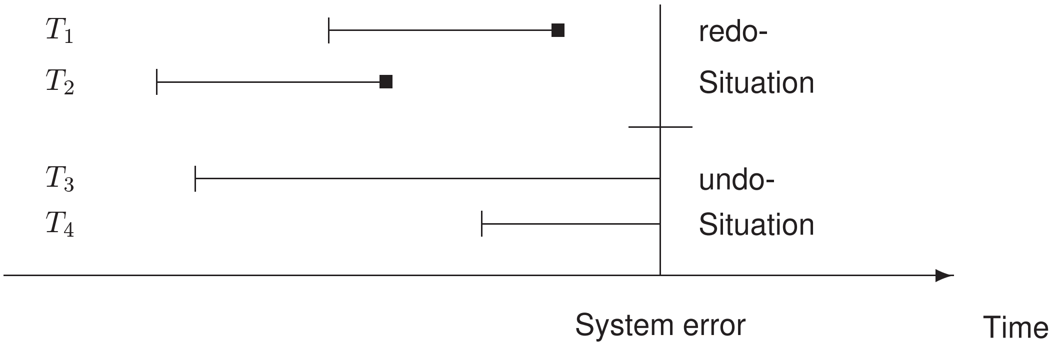
6.6 Safety: Error Recovery

What (dangerous) errors can happen to a database system?

- **Transaction errors**
  local, application-semantical error situations
  - error situation in the application program
  - user-initiated abort of transaction
  - violation of system restrictions (authentication etc)
  - resolving of a deadlock by aborting a transaction.
- **System errors**
  runtime environment crashes completely
  - hardware errors (main memory, processor)
  - faulty values in system tables that cause a software crash
- **Media crashes**
  database backend crashes
  crash of secondary memory (disk head errors ...)

Assumption: Transactions satisfy **strict 2PL**.

REDO- AND UNDO-SITUATIONS



- **redo-Situation:**
  A transaction has committed, and an error occurs.
- **undo-Situation:**
  A transaction already writes to the database before committing. During execution, an error occurs.

  undo-avoiding:

  write actions are only written to the database (“materialized”) *after* committing. Before, they are cached in a special buffer.

Database Log

The database system maintains a **log** (journal) where all changes in the database and all state changes of transactions (BOT/EOT) are recorded.

Entries (sequential):

(1) at begin of transaction: $(T, begin)$

(2) if a transaction T executes WX:

$$(\ T\ ,\ X\ ,\quad X_{old}\ ,\quad X_{new}\ )$$

value of X written by T **(after image)**

value of X before WX **(before image)**

(3) at commit: $(T, commit)$

(4) at abort: $(T, abort)$

Example 6.11
(write-immediate, no undo-avoiding)

T_1:	LA RA		WA CO UA				
T_2:		LB RB		LA RA WB		CO UA UB	
T_3:					LC RC		WC
Database:							
A:		A_0	$f_1(A_0)$				
B:		B_0		$f_2(f_1(A_0), B_0)$			
C:		C_0					$f_3(C_0)$
Buffers:							
T_1:			$A : f_1(A_0)$				
T_2:				$B : f_2(f_1(A_0), B_0)$			
T_3:							$C : f_3(C_0)$

Log:
$(T_1, A, A_0, f_1(A_0)), (T_1, CO), (T_2, B, B_0, f_2(f_1(A_0), B_0)), (T_2, CO), (T_3, C, C_0, f_3(C_0))$ □

Transaction Errors

Consider a transaction T that is aborted before reaching its COMMIT phase.

If undo-avoiding is used, no error handling is required (simply discard its log entries),

Otherwise, process log file backwards up to $(T, begin)$ and materialize for every entry (T, X, X_{old}, X_{new}) the (before-)value X_{old} for X in the database.

Note that due to strict 2PL, no other transaction could read values that have been written by T!

System Errors

Restart-Algorithm (without savepoints, for strict 2PL)

- $redone := \emptyset$ and $undone := \emptyset$.
- process the logfile backwards until end, or $redone \cup undone$ contains all database objects.

  For every entry (T, X, X_{old}, X_{new}):

  If $X \notin redone \cup undone$:

  - If the logfile contains $(T, commit)$ (then redo), then write X_{new} into the database and set $redone := redone \cup \{X\}$.
  - Otherwise (undo) write X_{old} into the database and set $undone := undone \cup \{X\}$. ("undo once" only correct for **strict 2PL**!)

  If undo-avoiding is used, no undo is required.

Example 6.12

Consider again Example 6.11.

(write-immediate, no undo-avoiding)					*Sys.error*	*state after restart*
T_1:	*LA RA*	*WA CO UA*				
T_2:	*LB RB*		*LA RA WB*		*CO UA UB*	
T_3:				*LC RC*	*WC*	
Database:						
A:	A_0	$f_1(A_0)$				$f_1(A_0)$
B:	B_0		$f_2(f_1(A_0), B_0)$			$f_2(f_1(A_0), B_0)$
C:	C_0				$f_3(C_0)$	C_0
Buffers:						
T_1:		$A : f_1(A_0)$				
T_2:			$B : f_2(f_1(A_0), B_0)$			
T_3:					$C : f_3(C_0)$	

Log:

$(T_1, A, A_0, f_1(A_0)), (T_1, CO), (T_2, B, B_0, f_2(f_1(A_0), B_0)), (T_2, CO), (T_3, C, C_0, f_3(C_0))$ □

Logging Requirements

Log granularity:

the log-granularity must be finer than the lock granularity. Otherwise, redo or undo can also delete effects of other transactions than intended.

Write-ahead:

before a write action is materialized in the database, it must be materialized in the log file (materialized means that it must actually be written to the DB, not only to a buffer – which could be lost)

Savepoints

... processing the log backwards ...

until the most recent **savepoint**.

Generation of a Savepoint

- Do not begin any transaction, and wait for all transactions to finish (COMMIT or ABORT).
- Materialize all changes in the database (force write caches).
- write *(checkpoint)* to the logfile

Media Crash

Solution: Redundancy

Strategy 1: keep a complete copy of the database (incl log)

Probability that both are destroyed at the same time is low (keep them in different computers in different buildings ...)

Writing of a tuple to the database means to write it also in the copy. Copy is written only after write to original is confirmed to be successful (otherwise e.g. an electrical breakdown kills both).

Strategy 2: periodical generation of an archive database (dump).

After generation of the dump, $(archive)$ is written to the logfile.

In case of a media crash, restart as follows:

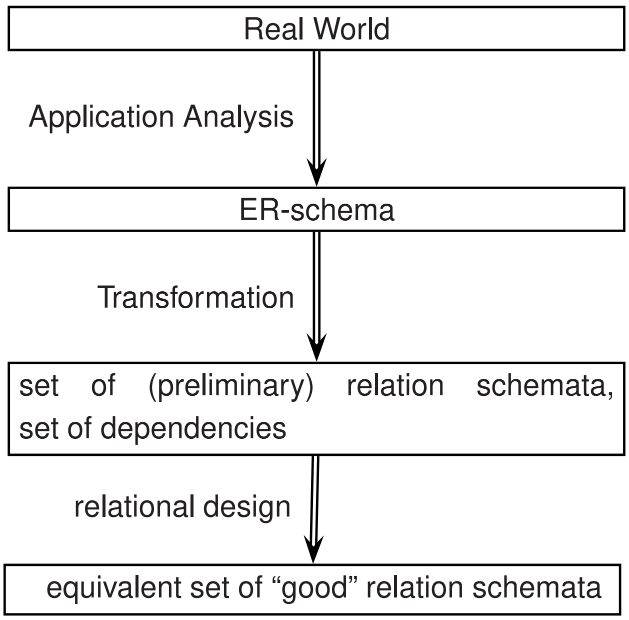
- Load the dump.
- apply restart-algorithm only wrt. redo of completed (committed) transactions back to the $(archive)$ entry.

Chapter 7
Design Theory of the Relational Model

Goal: a relational schema that suitably represents an excerpt of the real world.

- Real world implies integrity constraints (we have seen e.g. keys and referential integrity as *relational concepts*)
- Base of such concepts: *data dependencies*
- Representation must cope with these dependencies (from this design, we will *obtain* keys and referential integrity).

DESIGN STEPS



Motivation

Example 7.1
Consider the following situation: a supplier has contracts with several customers to deliver products regularly. For each product, the number of delivered items and the price is relevant.

Shipments

Name	*Address*	*Product*	*Number*	*Price*
Meier	*Stuttgart*	*Pizza*	*10*	*5.00*
Meier	*Stuttgart*	*Lasagne*	*15*	*6.00*
Meier	*Stuttgart*	*Salad*	*20*	*3.00*
Müller	*Freiburg*	*Pizza*	*12*	*5.00*
Müller	*Freiburg*	*Salad*	*15*	*3.00*

- *Redundancy*
- *caused problems:*

  (1) anomalies when updating or inserting (potential inconsistencies),

  (3) anomalies when deleting (delete Meier $\rightarrow$ information about price of Lasagne is lost)

Example 7.1 (Continued)
Refined Schema:

Customer

Name	*Address*
Meier	*Stuttgart*
Müller	*Freiburg*

Product

Product	*Price*
Pizza	*5.00*
Lasagne	*6.00*
Salad	*3.00*

Shipment′

Name	*Product*	*Number*
Meier	*Pizza*	*10*
Meier	*Lasagne*	*15*
Meier	*Salad*	*20*
Müller	*Pizza*	*12*
Müller	*Salad*	*15*

is the refined schema "better"?

- *is it equivalent?*
- *anomalies removed?* □

Required Notions

1. Analysis of relevant dependencies
2. criterion when to decompose a relation schema (and when a decomposition is equivalent) (based on (1))
3. measure for "quality" of a schema (in terms of (1))

7.1 Functional Dependencies

- Data dependencies that describe a **functional** relationship.

Let $\bar{V}$ a set of attributes and $r \in \mathrm{Rel}(\bar{V})$, $\bar{X}, \bar{Y} \subseteq \bar{V}$.
r satisfies the **functional dependency (FD)** $\bar{X} \to \bar{Y}$ if for all $t, s \in r$,

$$t[\bar{X}] = s[\bar{X}] \Rightarrow t[\bar{Y}] = s[\bar{Y}] \; .$$

For $\bar{Y} \subseteq \bar{X}$, $\bar{X} \to \bar{Y}$ is a **trivial** dependency (satisfied by every relation $r \in \mathrm{Rel}(\bar{V})$).

Refined Definition of "Relation Schema"

A **relation schema** $R(\bar{X}, \Sigma_{\bar{X}})$ consists of a name (here, R) and a finite set $\bar{X} = \{A_1, \ldots, A_m\}$, $m \geq 1$ of attributes:

- $\bar{X}$ is the **format** of the schema.
- $\Sigma_{\bar{X}}$ is a set of functional dependencies over $\bar{X}$.

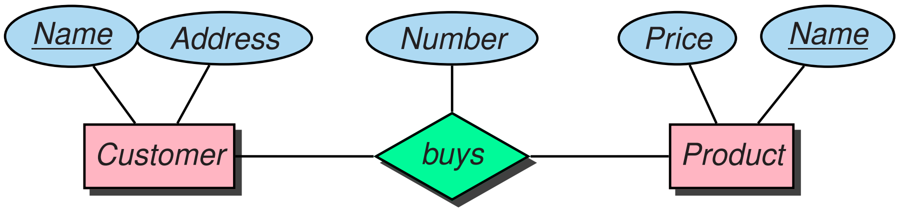
A relation $r \in \mathrm{Rel}(\bar{X})$ is an **instance** of R if it satisfies all dependencies in $\Sigma_{\bar{X}}$.
The set of all instances of R is denoted by $\mathrm{Sat}(\bar{X}, \Sigma_{\bar{X}})$.

Example 7.2
Consider again Example 7.1.

The given instance is in $Sat(\bar{X}, \Sigma_{\bar{X}})$ *for the following set* $\Sigma_{\bar{X}}$ *of FDs:*

Name $\rightarrow$ *Address*
Product $\rightarrow$ *Price*
(Name, Product) $\rightarrow$ *Number*

"Intuitive" ER-model of the problem:



□

7.1.1 Decomposition Based on Functional Dependencies

- Guarantees a "good" ER-model already all desirable properties of the relational model?

NO

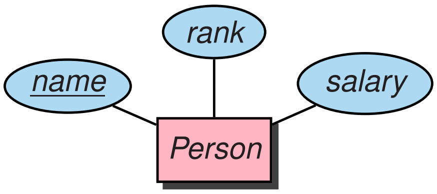
- is a formal dependency analysis necessary?

YES

- theory: based on *normal forms* of relational schemata

Analysis of Entity Sets

Example 7.3 (FDs of entity attributes)
Consider a staff database in a university. Persons (professors and lecturers) have names, ranks, and salaries.



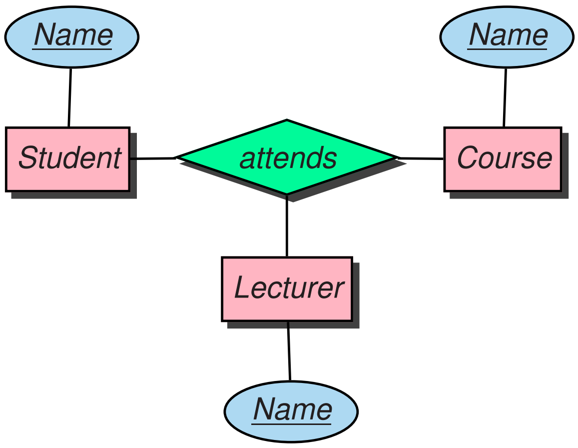
Person		
<u>Name</u>	Rank	Salary
G	full prof.	5000
T	full prof.	5000
S	associate prof.	4000
W	assistant	3000
P	assistant	3000

There is a functional dependency Rank $\rightarrow$ Salary.

Refined schema: *Person(<u>Name</u>, Rank)* □
SalaryTable(<u>Rank</u>, Salary)

Analysis of Relationship Sets

Example 7.4 (FDs of ternary relationships)
Students attend courses that are given by lecturers.



attends		
Student	Course	Lecturer
DStud1	Databases I	GL
DStud2	Databases I	GL
DStud2	Databases Internet	GL
ACSStud1	Databases Internet	GL
ACSStud1	Databases & SQL	WM
ACSStud2	Databases & SQL	WM
DStud1	Databases & SQL	WM

There is a functional dependency Course $\rightarrow$ Lecturer.

Refined schema: *reads(<u>Course</u>, Lecturer)* □
attends'(<u>Student</u>, <u>Course</u>)

7.1.2 Functional Dependency Theory

Let $R(\bar{V}, \mathcal{F})$ a relation schema where $\bar{X}, \bar{Y} \subseteq \bar{V}$, and $\mathcal{F}$ is a set of functional dependencies over $\bar{V}$.

Definition 7.1

- *$\mathcal{F}$ **implies** a functional dependency $\bar{X} \to \bar{Y}$, written as $\mathcal{F} \models \bar{X} \to \bar{Y}$, if and only if every relation $r \in$ Sat$(\bar{V}, \mathcal{F})$ satisfies $\bar{X} \to \bar{Y}$.*
- *$\mathcal{F}^+ = \{\bar{X} \to \bar{Y} \mid \mathcal{F} \models \bar{X} \to \bar{Y}\}$ is the **closure** of $\mathcal{F}$.* □

Definition 7.2

*Let $\bar{V} = \{A_1 \ldots A_n\}$. $\bar{X}$ is a **key** of $\bar{V}$ (wrt. $\mathcal{F}$) if and only if*

- $\bar{X} \to A_1 \ldots A_n \in \mathcal{F}^+$,
- $\bar{Y} \subsetneq \bar{X} \Rightarrow \bar{Y} \to A_1 \ldots A_n \notin \mathcal{F}^+$.

For a key $\bar{X}$, each $\bar{Y} \supseteq \bar{X}$ is a superkey. □

For an attribute A such that $A \in \bar{X}$ for any key $\bar{X}$, A is a **key attribute.** If there is no key $\bar{X}$ such that $A \in \bar{X}$, then A is a **non-key attribute.**

Closure of FDs

Problem: How to decide whether $\bar{X} \to \bar{Y} \in \mathcal{F}^+$? (Membership Test)

The test is based on the **Armstrong-Axioms**:

Let $\mathcal{F}$ a set of FDs over $\bar{V}$ and $r \in$ Sat$(\bar{V}, \mathcal{F})$.

(A1)	Reflexivity: If $\bar{Y} \subseteq \bar{X} \subseteq \bar{V}$, then r satisfies $\bar{X} \to \bar{Y}$.
(A2)	Augmentation: If $\bar{X} \to \bar{Y} \in \mathcal{F}$ and $\bar{Z} \subseteq \bar{V}$, then r satisfies $\overline{XZ} \to \overline{YZ}$.
(A3)	Transitivity: If $\bar{X} \to \bar{Y}$ and $\bar{Y} \to \bar{Z} \in \mathcal{F}$, then r satisfies $\bar{X} \to \bar{Z}$.

The Armstrong-Axioms can be used as inference rules for FDs.

Theorem 7.1

*The Armstrong-Axioms are **correct**, i.e., all derived FDs are in $\mathcal{F}^+$, and they are **complete**, i.e., all FDs in $\mathcal{F}^+$ can be derived.* □

CLOSURE OF FDS (CONT'D)

Armstrong Axioms can especially be used for searching which attributes depend on a given $\bar{X} \subseteq V$.

Definition 7.3
For $\bar{X} \subseteq \bar{V}$, $\bar{X}^+$ is the set of all $A \in \bar{V}$ such that $\bar{X} \rightarrow A$ can be derived by the Armstrong axioms. $\bar{X}^+$ is called the ***(Attribute-)closure*** *of $\bar{X}$ (wrt. $\mathcal{F}$).* □

Exercise 7.1
Consider a relation schema $R(\bar{V}, \mathcal{F})$ such that $\bar{K}$ is a key. What is $\bar{K}^+$? □

MEMBERSHIP PROBLEM

Check whether $\bar{X} \rightarrow \bar{Y} \in \mathcal{F}^+$?

Variant 1 :
Compute $\mathcal{F}^+$ from $\mathcal{F}$ using (A1)–(A3) until either $\bar{X} \rightarrow \bar{Y}$ is derived, or the process stops. Then, we have $\mathcal{F}^+$, and $\bar{X} \rightarrow \bar{Y} \notin \mathcal{F}^+$.

This algorithm is not efficient, since it has (systematically applied) at least the time complexity $O(2^{||\mathcal{F}||})$.

Example 7.5
Consider $\bar{V} = \{A, B_1, \ldots, B_n, C, D\}$ with $\mathcal{F} = \{A \rightarrow B_1, \ldots, A \rightarrow B_n\}$. Then, $A \rightarrow \bar{Y} \in \mathcal{F}^+$ for all $\bar{Y} \subseteq \{B_1, \ldots, B_n\}$. Thus, computation of $\mathcal{F}$ needs to compute 2^n items (before a negative answer for any other FD – e.g. the question whether $C \rightarrow D$ holds – can be stated). □

An Equivalent Set of Rules

Lemma 7.1
Consider a relation schema $R(\bar{V}, \mathcal{F})$ such that $A \in \bar{V}$ and $\bar{X}, \bar{Y}, \bar{Z}, \bar{W} \subseteq \bar{V}$, and $\mathcal{F}$ is a set of functional dependencies over $\bar{V}$, and $r \in$ Sat$(\bar{V}, \mathcal{F})$. Then:

- **(A4)** *Union: If $\bar{X} \to \bar{Y}$ and $\bar{X} \to \bar{Z} \in \mathcal{F}$, then r satisfies $\bar{X} \to \overline{YZ}$.*
- **(A5)** *Pseudo-transitivity: If $\bar{X} \to \bar{Y}$ and $\overline{WY} \to \bar{Z} \in \mathcal{F}$, then r satisfies $\overline{XW} \to \bar{Z}$.*
- **(A6)** *Decomposition: If $\bar{X} \to \bar{Y} \in \mathcal{F}$ and $\bar{Z} \subseteq \bar{Y}$, then r satisfies $\bar{X} \to \bar{Z}$.*
- **(A7)** *Reflexivity: r satisfies $\bar{X} \to \bar{X}$*
- **(A8)** *Accumulation: If $\bar{X} \to \overline{YZ}$ and $\bar{Z} \to \overline{AW} \in \mathcal{F}$, then r satisfies $\bar{X} \to \overline{YZA}$.* □

Lemma 7.2
The rule sets $\{$(A1), (A2), (A3)$\}$ and $\{$(A6), (A7), (A8)$\}$ are equivalent, i.e., for given $\mathcal{F}$, the same set of FDs can be derived. □

. . . and $\{$(A6), (A7), (A8)$\}$ leads to a more efficient algorithm!

Membership Problem Revisited

Check whether $\bar{X} \to \bar{Y} \in \mathcal{F}^+$?

Variant 2 :
Compute $\bar{X}^+$ and check if $\bar{Y} \subseteq \bar{X}^+$.

$\bar{X}^+$-Algorithm:

```
result := X̄;      /* (A7) */
WHILE (changes to result) DO
  FOR each W̄ → Z̄ ∈ F DO      /* (A8) */
     IF (W̄ ⊆ result ) THEN result := result ∪ Z̄ ;
  end;
check if Ȳ ⊆ result      /* (A6) */;
```

Theorem 7.2
The $\bar{X}^+$-algorithm computes $\bar{X}^+$ and terminates. Its time complexity is $O(||\mathcal{F}||^2)$.
There is an optimized variant in $O(||\mathcal{F}||)$. □

Example 7.6

$\mathcal{F} = \{AB \to E, BE \to I, E \to G, GI \to H\}$, *check if* $AB \to GH \in \mathcal{F}^+$ *?*

Derivation by (A7)–(A8)		*Intermediate result $\bar{X}_i$ of the $\bar{X}^+$-algorithm*
(A7)	$AB \to AB$	$\bar{X}_0 = \{A, B\}$
(A8)	$[\, AB \to E \,]$	
	$AB \to ABE$	$\bar{X}_1 = \{A, B, E\}$
(A8)	$[\, BE \to I \,]$	
	$AB \to ABEI$	$\bar{X}_2 = \{A, B, E, I\}$
(A8)	$[\, E \to G \,]$	
	$AB \to ABEIG$	$\bar{X}_3 = \{A, B, E, I, G\}$
(A8)	$[\, GI \to H \,]$	
	$AB \to ABEIGH$	$\bar{X}_4 = \{A, B, E, I, G, H\}$
final step with (A6):		
(A6)	$AB \to GH$	

□

DETERMINING A KEY

Consider a relation schema $R = (\bar{V}, \mathcal{F})$.

- The $\bar{X}^+$-algorithm allows for determining a key of R in time $O(|\bar{V}|\ ||\mathcal{F}||)$:
  Start with the superkey $\bar{V}$ and try to delete attributes as long as the closure of the remaining attributes is still the whole $\bar{V}$. If no more attributes can be deleted, a key is found.
- In the general case, it is not possible to determine *all* keys of a relation schema efficiently. Note that the problem "is there a key with at most k attributes?" is NP-complete.

Aside: Unique Keys

Theorem 7.3
Let $\mathcal{F} = \{\bar{X}_1 \to \bar{Y}_1, \ldots, \bar{X}_p \to \bar{Y}_p\}$.
Let $\bar{Z}_i = \bar{Y}_i \setminus \bar{X}_i$ for $1 \leq i \leq p$.

$R(\bar{V})$ has a unique key if and only if $\bar{V} \setminus (\bar{Z}_1 \cup \ldots \cup \bar{Z}_p)$ is a superkey.
(note that $\bar{K}$ is a superkey if $\bar{K}^+ = \bar{V}$). □

Proof: Assume $\bar{K}$ to be the unique key of R. Then, $\bar{K}$ is contained in every superkey. For each $1 \leq i \leq p$, $\bar{V} \setminus \bar{Z}_i$ is a superkey (since $\bar{Z}_i$ is determined by $\bar{X}_i$).

Thus, $\bar{K} \subseteq \cap_{i=1}^{p}(\bar{V} \setminus \bar{Z}_i)$. The right side is equivalent to $\bar{V} \setminus (\bar{Z}_1 \cup \ldots \cup \bar{Z}_p)$. Thus, $\bar{V} \setminus (\bar{Z}_1 \cup \ldots \cup \bar{Z}_p)$ is a superkey (of $\bar{K}$).

Assume $\bar{K} = \bar{V} \setminus (\bar{Z}_1 \cup \ldots \cup \bar{Z}_p)$ a superkey. We show that $\bar{K}$ is contained in every superkey, and thus it is the only key. Suppose a superkey $\bar{L}$ such that there is an attribute $A \in \bar{K} \setminus \bar{L}$. Then, $A \notin \bar{L}^+$ (since it is not in any of the $\bar{Z}_i$). Thus, L is not a superkey (since $\bar{L}^+ \subsetneq \bar{V}$) – contradiction.

Aside: Sets of FDs

Consider sets $\mathcal{F}, \mathcal{G}$ of functional dependencies. $\mathcal{F}, \mathcal{G}$ are **equivalent** if and only if $\mathcal{F}^+ = \mathcal{G}^+$.

Definition 7.4
*$\mathcal{F}$ is **minimal** if and only if*

1. *For every $\bar{X} \to \bar{Y} \in \mathcal{F}$, $\bar{Y}$ consists of a single attribute,*
2. *For every $\bar{X} \to A \in \mathcal{F}$, $\mathcal{F} \setminus \{\bar{X} \to A\}$ is not equivalent to $\mathcal{F}$,*
3. *If $\bar{X} \to A \in \mathcal{F}$ and $\bar{Z} \subset \bar{X}$, then $\mathcal{F} \setminus \{\bar{X} \to A\} \cup \{\bar{Z} \to A\}$ is not equivalent to $\mathcal{F}$.* □

Theorem 7.4
For each set $\mathcal{F}$ of functional dependencies, there is an equivalent minimal set $\mathcal{F}^{min}$ of functional dependencies. □

Remark:

- $\mathcal{F}^{min}$ can be computed by the $\bar{X}^+$-algorithm (without computing $\mathcal{F}^+$) in polynomial time.
- $\mathcal{F}^{min}$ is not necessarily unique.

7.2 Decomposition of Relation Schemata

In Example 7.1 (Slide 205), a relation has been *decomposed* for yielding a better behavior.

Definition 7.5

- *Let $\bar{V}$ a set of atributes, and $\mathcal{F}$ a set of FDs to $\bar{V}$. Then, $\rho = \{\bar{X}_1, \ldots, \bar{X}_n\}$ s.t. $\bar{X}_1 \cup \ldots \cup \bar{X}_n = \bar{V}$ and for each i, $\bar{X}_i \subseteq \bar{V}$ is a **decomposition** of $\bar{V}$.* □

Example 7.7

Consider again Example 7.1. There, $\bar{V} = \{$Name, Address, Product, Number, Price$\}$.

$\rho = \{\{$Name, Address$\}, \{$Product, Price$\}, \{$Name, Product, Number$\}\}$. □

Properties of Decompositions

Losslessness: The complete tuples must be reconstructable by joining the decomposed relations without getting additional tuples that have not been there originally.

Example 7.8

Consider again Example 7.4, now with a decomposition into hears(Student,Lecturer) *and* attends'(Student, Course).

Then, the join hears $\bowtie$ attends' *yields a tuple (*DStud1,Databases&SQL,GL*).* □

Definition 7.6

Consider a relation schema $R(\bar{V}, \mathcal{F})$ and a decomposition $\rho = \{\bar{X}_1, \ldots, \bar{X}_n\}$ of R.

*ρ is **lossless** if and only if for every relation $r \in$ Sat$(\bar{V}, \mathcal{F})$,*

$$r = \pi[\bar{X}_1](r) \bowtie \ldots \bowtie \pi[\bar{X}_k](r) \ .$$ □

Properties of Decompositions (Cont'd)

dependency-preservation: the dependencies can be tested using the decomposed tables only, i.e., without having to recompute the join.

Definition 7.7
Consider a relation schema $R(\bar{V}, \mathcal{F})$ and a decomposition $\rho = \{\bar{X}_1, \ldots, \bar{X}_n\}$ of R.
$\pi[Z](\mathcal{F}) = \{X \to Y \in \mathcal{F}^+ \mid XY \subseteq Z\}$ is the projection of $\mathcal{F}$ to Z.
*ρ is **dependency-preserving** if and only if for all i,*

$$\bigcup_{i=1}^{n} \pi[\bar{X}_i](\mathcal{F}) \equiv \mathcal{F} \; .$$

□

Dependency-preservation means that FDs can be distributed over the decomposition without losing anything.

We will first discuss losslessness.

7.2.1 Lossless Decompositions

- We cannot lose tuples by (wrong) decompositions, but "information" about relationships.

Example 7.9
Consider again Examples 7.4 and 7.8.

1. *We have attends* $= \underbrace{\pi[\textit{Course}, \textit{Lecturer}](\textit{attends})}_{\textit{reads}} \bowtie \underbrace{\pi[\textit{Student}, \textit{Course}](\textit{attends})}_{\textit{attends}'}$
2. *and attends* $\subsetneq \underbrace{\pi[\textit{Student}, \textit{Lecturer}](\textit{attends})}_{\textit{hears}} \bowtie \underbrace{\pi[\textit{Student}, \textit{Course}](\textit{attends})}_{\textit{attends}'}$ □

Lemma 7.3
Consider a relation $r \in$ Rel$(\bar{V})$ and a decomposition $\rho = \{\bar{X}_1, \ldots, \bar{X}_k\}$ of $\bar{V}$.

Then,

$$r \subseteq \pi[\bar{X}_1](r) \bowtie \ldots \bowtie \pi[\bar{X}_k](r) \; .$$

□

TEST FOR LOSSLESSNESS (CHASE-ALGORITHM FOR FDS)

Input: a relation schema $R(\bar{V}, \mathcal{F})$, where $\bar{V} = \{A_1, \ldots, A_n\}$, $\rho = \{\bar{X}_1, \ldots, \bar{X}_k\}$.

Algorithm:
Construct a table T with n columns and k rows. Column j stands for A_j, row i for $\bar{X}_i$ as follows:

- $T_{(i,j)} = a_j$ if $A_j \in \bar{X}_i$; otherwise $T_{(i,j)} = b_{ij}$.
- As long as T changes, do
  Consider a FD $\bar{X} \to \bar{Y} \in \mathcal{F}$. If there are rows $z_1, z_2 \in T$ which coincide for all $\bar{X}$-columns, but not in all $\bar{Y}$-columns, then make $\bar{Y}$-values the same:
  - For each $\bar{Y}$-column j:
  - if one of the symbols is a_j, then replace the other symbol in the whole column by a_j.
  - if both symbols are of the form b_{ij}, then replace arbitrarily one of them by the other in the whole column

  Note: The algorithm corresponds to *enforcing* the FDs.

Result: ρ ist lossless if and only if $(a_1, \ldots, a_n) \in T$.

Note: the final table satisfies all FDs in $\mathcal{F}$.

Theorem 7.5
The above algorithm for testing losslessness is correct. □

Example 7.10 (Chase)
$\bar{V} = ABCDE$, $\rho = (AD, AB, BE, CDE, AE)$;
$\mathcal{F} = \{A \to C, B \to C, C \to D, DE \to C, CE \to A\}$

A	B	C	D	E
a_1	b_{12}	b_{13}	a_4	b_{15}
a_1	a_2	b_{23}	b_{24}	b_{25}
b_{31}	a_2	b_{33}	b_{34}	a_5
b_{41}	b_{42}	a_3	a_4	a_5
a_1	b_{52}	b_{53}	b_{54}	a_5

A	B	C	D	E
a_1	b_{12}	a_3	a_4	b_{15}
a_1	a_2	a_3	a_4	b_{25}
a_1	a_2	a_3	a_4	a_5
a_1	b_{42}	a_3	a_4	a_5
a_1	b_{52}	a_3	a_4	a_5

□

Corollary 7.1

Consider a set $\bar{V}$ *of attributes, a set* $\mathcal{F}$ *of functional dependencies, and a decomposition* $\rho = \{\bar{X}_1, \bar{X}_2\}$ *of* $\bar{V}$. ρ *is lossless if and only if*

$$(\bar{X}_1 \cap \bar{X}_2) \to (\bar{X}_1 \setminus \bar{X}_2) \in \mathcal{F}^+, \text{ or } (\bar{X}_1 \cap \bar{X}_2) \to (\bar{X}_2 \setminus \bar{X}_1) \in \mathcal{F}^+ .$$

□

Example 7.11

Consider again Examples 7.4, 7.8 and 7.9 where we used the schema

$$\textit{attends}((\textit{Student, Course, Lecturer}), \{\textit{Course} \to \textit{Lecturer}\})$$

- $\rho_1 = \{\{\textit{Course}, \textit{Lecturer}\}, \{\textit{Student}, \textit{Course}\}\}$ *is lossless.*
- $\rho_2 = \{\{\textit{Student}, \textit{Lecturer}\}, \{\textit{Student}, \textit{Course}\}\}$ *is not lossless.*

□

General conclusion for ternary relations:

- for any (useful) decomposition into two binary relations, there is one attribute A that is contained in both relations.
- the decomposition is lossless if at least one of the other attributes is functionally dependent only on A.

In the above example, we had the functional dependency *Course* $\to$ *Lecturer* which made the decomposition possible.

7.2.2 Dependency-Preservation

Example 7.12
Consider again Examples 7.1 and 7.7 where we used the schema

$$\textit{Shipments}(\ \{\textit{Name}, \textit{Address}, \textit{Product}, \textit{Number}, \textit{Price}\},$$
$$\{\textit{Name} \to \textit{Address},\ \textit{Product} \to \textit{Price},\ (\textit{Name}, \textit{Product}) \to \textit{Number}\})$$

and the decomposition

$$\rho = \{\{\textit{Name}, \textit{Address}\},\ \{\textit{Product}, \textit{Price}\},\ \{\textit{Name}, \textit{Product}, \textit{Number}\}\}\ .$$

Recall that $\pi[Z](\mathcal{F}) = \{X \to Y \in \mathcal{F}^+ \mid XY \subseteq Z\}$

$\pi[\textit{Name, Address}](\mathcal{F}) \supseteq \{\textit{Name} \to \textit{Address}$
$\pi[\textit{Product, Price}](\mathcal{F}) \supseteq \{\textit{Product} \to \textit{Price}$
$\pi[\textit{Name, Product, Number}](\mathcal{F}) \supseteq \{(\textit{Name,Product}) \to \textit{Number}$

So, all FD's immediately survive. □

Another, abstract Example

Example 7.13
$V = \{A, B, C, D\}$, $\rho = \{AB, BC\}$
$\mathcal{F} = \{A \to B, B \to C, C \to A\}$

ρ is dependency-preserving (check whether $C \to A$ is preserved).

Recall that $\pi[Z](\mathcal{F}) = \{X \to Y \in \mathcal{F}^+ \mid XY \subseteq Z\}$

$\pi[AB](\mathcal{F}) \supseteq \{A \to B, B \to A\}$
$\pi[BC](\mathcal{F}) \supseteq \{B \to C, C \to B\}$
$C \to A \in (\pi[AB](\mathcal{F}) \cup \pi[BC](\mathcal{F}))^+$ □

DEPENDENCY-PRESERVATION

There are lossless decompositions that are not dependency-preserving:

Example 7.14
Consider $R = (\bar{V}, \mathcal{F})$*, where* $\bar{V} = \{$*City, Address, Zip*$\}$*, and*
$\mathcal{F} = \{($*City,Address*$) \to$ *Zip*, *Zip* $\to$ *City*$\}$.

The decomposition R_1(<u>Address, Zip</u>) *and* R_2(City, <u>Zip</u>) *is lossless since*
$(R_1 \cap R_2) \to (R_2 \setminus R_1) \in \mathcal{F}$*, but is not dependency-preserving.*

(note that the keys of R *are (*Address, Zip*) and (*City, Address*).)*

R	City	Address	Zip
	FR	Herdern	79106
	FR	Flughafen	79110
	FR	Mooswald	79110
	S	Flughafen	70629

R_1	Address	Zip
	Herdern	79106
	Flughafen	79110
	Mooswald	79110
	Flughafen	70629

R_2	City	Zip
	FR	79106
	FR	79110
	S	70629

*Insert (*FR,Herdern,79100*) and check the FDs.* □

... and now to a systematic characterization:

7.3 Normal Forms based on FDs

Task:

Consider a schema $R = (\bar{V}, \mathcal{F})$. Find a decomposition $\rho = (\bar{X}_1, \ldots, \bar{X}_n)$ of R such that

1. each $R_i = (\bar{X}_i, \pi[\bar{X}_i](\mathcal{F}))$, $1 \leq i \leq n$ ist in some normal form,
2. ρ is lossless and (if possible) dependency-preserving,
3. n is minimal.

Non-normalized Data

Nested Relations:

Nested_Languages			
Code	**Name**	**Languages**	
D	Germany	German	100
CH	Switzerland	German	65
		French	18
		Italian	12
⋮	⋮	⋮	

Non-atomic values:

Products		
Code	**GDP**	**Products**
D	1452200	steel, coal, chemicals, machinery, vehicles
CH	158500	machinery, chemicals, watches
⋮	⋮	⋮

1ST NORMAL FORM (1NF)

Definition 7.8
A relation schema is in 1NF if and only if its attribute domains are atomic. □

Non-normalized relations are transformed into 1NF by expanding groups.

Note that functional dependencies arise.

Example 7.15

Languages			
Code	**Name**	**Language**	**Percent**
D	*Germany*	*German*	*100*
CH	*Switzerland*	*German*	*65*
CH	*Switzerland*	*French*	*18*
CH	*Switzerland*	*Italian*	*12*
⋮	⋮	⋮	⋮

$$\mathcal{F} = \{Code \to Name,\ Name \to Code,\ (Code, Language) \to Percent,\ (Name, Language) \to Percent\}$$ □

Example 7.16

Economy		
Code	**GDP**	**Product**
D	*1452200*	*steel*
D	*1452200*	*coal*
D	*1452200*	*chemicals*
D	*1452200*	*machinery*
D	*1452200*	*vehicles*
CH	*158500*	*machinery*
CH	*158500*	*chemicals*
CH	*158500*	*watches*
⋮	⋮	⋮

$\mathcal{F} = \{$*Code* $\rightarrow$ *GDP*$\}$ □

3rd Normal Form (3NF)

- In Example 7.16, the GDP information (e.g., (*D, 1452200*)) is stored redundantly.

Definition 7.9
A relation schema $R = (\bar{V}, \mathcal{F})$ is in 3NF if and only if for each non-key *attribute A:*

- *For each $\bar{X} \rightarrow A \in \mathcal{F}$ such that $A \notin \bar{X}$, $\bar{X}$ contains a key.* □

3NF forbids non-trivial FDs, where a non-key attribute A is functionally dependent on some $\bar{X}$ that is a proper subset of a key. Such FDs cause redundancy.

Example 7.17
Consider again Example 7.16: GDP *is not a key attribute. It depends on* Code*, but* Code *does not contain a key.*

Split the Economy *relation into relations* Economy'(<u>Code</u>,GDP) *and* Products(<u>Code</u>,<u>Product</u>)*.* □

3NF-Decomposition is always lossless and dependency-preserving.

Boyce-Codd Normal Form (BCNF)

- In Example 7.15 (Languages), the name (e.g., *D, Germany*) is stored redundantly. (Note that *Name* is a key attribute there – thus 3NF is not applicable.)

BCNF extends 3NF for key attributes:

Definition 7.10
A relation schema $R = (\bar{V}, \mathcal{F})$ is in BCNF if and only if for each attribute A:

- *For each $\bar{X} \to A \in \mathcal{F}$ such that $A \notin \bar{X}$, $\bar{X}$ contains a key.* □

Example 7.18
Consider again Example 7.15: Name *depends on* Code*, but* Code *does not contain a key.*

Split the Languages *relation into relations* Country(Code,Name) *and* Languages'(Code,Language,Percent)*.*

In this case, the decomposition is lossless and dependency-preserving. □

BCNF (Cont'd)

- BCNF-Decomposition is always lossless, but not necessarily dependency-preserving.

Example 7.19
Consider again Example7.14:

$R = (\bar{V}, \mathcal{F})$, *where* $\bar{V} = \{$*City, Address, Zip*$\}$, *and* $\mathcal{F} = \{($*City,Address*$) \to$ *Zip*, *Zip* $\to$ *City*$\}$.

R is in 3NF, but not in BCNF.

The decomposition R_1(Address, Zip) *and* R_2(City,Zip) *transforms it in a BCNF schema.*

We have shown that this decomposition is lossless, but not dependency-preserving. □

PROPERTIES OF BCNF AND 3NF

Theorem 7.6
If a relation schema R has exactly one key, then R is in BCNF if and only if R is in 3NF.

Proof: *Obviously, BCNF implies 3NF. Assume R in 3NF and $\bar{K}$ its only key. Assume a FD $\bar{X} \to A \in \mathcal{F}$.*

We show that $\bar{X} \to A$ is trivial (i.e., $A \in \bar{X}$). Since R is in 3NF, it is sufficient to consider the case where A is a key attribute.

$(\bar{K} - A) \cup \bar{X}$ is a superkey (since $\bar{X} \to A$ and A is part of $\bar{K}$). Thus, there is a key $\bar{K}' \subseteq (\bar{K} - A) \cup \bar{X}$. Since there is only a single key, $\bar{K} = \bar{K}'$. Thus, since $A \in \bar{K}$, also $A \in K'$ – thus it must be in $\bar{X}$. □

PROPERTIES OF BCNF AND 3NF (CONT'D)

Lemma 7.4
A relation schema $R = (\bar{V}, \mathcal{F})$ is in BCNF if and only if for each non-trivial FD $\bar{X} \to A \in \mathcal{F}^+$, $\bar{X}$ is a superkey.

Proof:

- *"if" is obvious.*
- *We show that if $\bar{X} \to A \in \mathcal{F}^+$ and $A \notin \bar{X}$, then $\bar{X} \to \bar{V} \in \mathcal{F}^+$.*

  We have $A \in \bar{X}^+ \setminus \bar{X}$. Thus, there is a non-trivial FD $\bar{Y} \to A \in \mathcal{F}$ that is used by the $\bar{X}^+$-algorithm for adding A to $\bar{X}^+$. For this, $\bar{Y} \subseteq \bar{X}^+$, i.e., $\bar{X} \to \bar{Y} \in \mathcal{F}^+$.

  Since R is in BCNF, $\bar{Y}$ is a superkey. Since $\bar{X} \to \bar{Y} \in \mathcal{F}^+$, $\bar{X}$ must already be a superkey – i.e., $\bar{X} \to \bar{V} \in \mathcal{F}^+$. □

Corollary 7.2
A relation schema $R = (\bar{V}, \mathcal{F})$ is in BCNF if and only if $R' = (\bar{V}, \mathcal{F}^+)$ is in BCNF. □

- Lemma 7.4 and Corollary 7.2 analogously hold for 3NF.

Practical Aspects

- BCNF can be tested in polynomial time.
  Sketch: Use the $\bar{X}^+$-algorithm for each FD $\bar{X} \to \bar{Y}$ to check if $\bar{X}$ is a superkey.
- Testing 3NF is NP-complete
  - polynomially check if BCNF – if "yes", OK
  - if "no", the check whether A is a key attribute is exponential.
- Consider a set $\mathcal{F}$ of FDs over $\bar{V}$, and $\bar{X} \subseteq \bar{V}$.
  Then, for computing $\pi[\bar{X}](\mathcal{F})$, only algorithms are known that are (in the worst case) exponential in $|\bar{X}|$.
  Sketch: For every $\bar{Y} \subseteq \bar{X}$, compute $\bar{Y}^+$ and add $\bar{Y} \to (\bar{Y}^+ \cap \bar{X})$ to $\pi[\bar{X}](\mathcal{F})^+$
  (no way to compute $\pi[\bar{X}](\mathcal{F})$ without the closure).

Practical Aspects (Cont'd)

Lemma 7.5
For a relation schema $R = (\bar{V}, \mathcal{F})$ s.t. there is a FD $\bar{X} \to \bar{Y}$ where $\bar{X} \cap \bar{Y} = \emptyset$, the decomposition $\rho = (R \setminus \bar{Y}, \overline{XY})$ is lossless. □

Proof *Proof: Use Corollary 7.1 (Slide 7.1):*
$(R \setminus \bar{Y}) \cap \overline{XY} = \bar{X}$, $\overline{XY} \setminus (R \setminus \bar{Y}) = \bar{Y}$*, and we have* $\bar{X} \to \bar{Y}$*.* □

... this can now be used for an algorithm

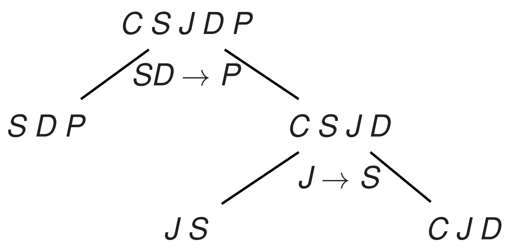
7.3.1 BCNF-Analysis: lossless, but not dependency-preserving

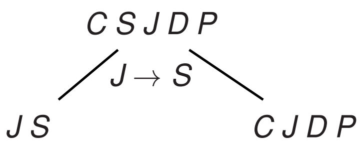
Input: a relation schema $R = (\bar{V}, \mathcal{F})$ that is not in BCNF.

Consider a FD $\bar{X} \to A \in \mathcal{F}$ that violates the BCNF condition.

- Decomposition of $\bar{V}$: $\rho = (\overline{XA}, \bar{V} - A)$ (A has been stored redundantly)
- $R_1 = (\overline{XA}, \pi[\overline{XA}](\mathcal{F}))$
- $R_2 = (\bar{V} - A, \pi[\bar{V} - A](\mathcal{F}))$,
- check whether R_1 and R_2 satisfy the BCNF condition, apply algorithm recursively.

Example 7.20
Let $\bar{V} = \{C,S,J,D,P\}$*,* $\mathcal{F} = \{SD \to P, J \to S\}$.





$SD \to P$ is not preserved. □

7.3.2 3NF-Analysis: lossless and dependency-preserving

- Sketch: BCNF – and repair.

Consider a relation schema $R = (\bar{V}, \mathcal{F})$ such that

- $\mathcal{F}$ is minimal, and
- $\rho = (\bar{X}_1, \ldots, \bar{X}_k)$ is a decomposition of $\bar{V}$ such that all schemata $R_i = (\bar{X}_i, \pi[\bar{X}_i](\mathcal{F}))$ are in BCNF.
  (possibly not dependency-preserving)
- For each such FD $\bar{X} \to A$ that is not preserved, extend ρ with $\overline{XA}$; the corresponding schema is $(\overline{XA}, \pi[\overline{XA}](\mathcal{F}))$.
- The resulting decomposition is obviously lossless and additionally dependency-preserving. Each of the new schemata is in 3NF.

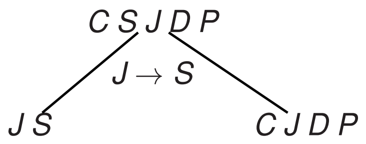
Proof Sketch: Since $\bar{X} \to A \in \mathcal{F}$ and $\mathcal{F}$ minimal, there is no $\bar{Y} \to A$ for any $\bar{Y} \subset \bar{X}$. Thus, $\bar{X}$ is a key for $\overline{XA}$ and all other FDs over $\overline{XA}$ are defined only over $\bar{X}$. Thus, they cannot violate the 3NF-condition (but the BCNF-condition).

Example 7.21

Consider again Example 7.20.

$\bar{V}$ = {C,S,J,D,P}, $\mathcal{F}$ = {SD $\rightarrow$ P, J $\rightarrow$ S}

- *The first decomposition is dependency-preserving.*
- *The second decomposition*



does not preserve SD $\rightarrow$ P.

The 3NF-analysis algorithm adds S D P. □

7.3.3 3NF-Synthesis: lossless and dependency-preserving

Input: relation schema $R = (\bar{V}, \mathcal{F})$ and $\mathcal{F}^{min}$.

1. Consider maximal sets of FDs from F^{min} with the same left hand side. Let $\{\bar{X} \rightarrow A_1,\ \bar{X} \rightarrow A_2,\ \ldots\}$ such a set.
   For every set, generate a schema with the format $\overline{XA_1A_2\ldots}$.
2. If none of the formats from (1) contains a key of R, take any key $\bar{K}$ of R and add a schema with format K.

- The 3NF-Synthese-Algorithm is polynomial in time.
- the resulting ρ ist not necessarily minimal:
  Consider $\bar{V} = \{AB\}$ with $\mathcal{F}^{min} = \{A \rightarrow B, B \rightarrow A\}$. Then, $\rho = (\underline{A}B, \underline{B}A)$.
- Recall that in contrast, it is NP-complete to check if a *given* schema is in 3NF.

Correctness

- Since we consider $\mathcal{F}^{min}$, the generated schemata are in 3NF.
- ρ is dependency-preserving since for every $\bar{X} \to \bar{Y} \in \mathcal{F}^{min}$, a format is generated that contains $\overline{XY}$.
- ρ ist lossless since ρ contains a key of the original schema. Using this tuple, in T^* (cf. Theorem 7.5) contains a row that consists of a_is:

  Consider the steps of the $\bar{X}^+$-algorithm that add – w.l.o.g. – the attributes $A_1, A_2, \ldots, A_k$ from $\bar{V} \setminus \bar{X}$ to $\bar{X}^+$. Show by induction that column of A_i in the row of $\bar{X}$ is set to a_i.
  - $i = 0$: nothing to show.
  - $i - 1 \to i$: A_i is added to $\bar{X}^+$ due to a FD $\bar{Y} \to A_i$ where $\bar{Y} \subseteq \bar{X} \cup \{A_1, \ldots, A_{i-1}\}$. Furthermore, $\overline{YA_i} \subseteq \bar{X}'$ for some $\bar{X}' \in \rho$ (generated by step (1)) and the rows of $\bar{X}$ and $\bar{X}'$ coincide for $\bar{Y}$ (only as). Then, the chase copies the a_i from the row of $\bar{X}'$ to the row of $\bar{X}$.

7.4 Join Dependencies and Multivalued Dependencies

Example 7.22

Consider the following Non-1NF table:

cco		
<u>Country</u>	<u>Continents</u>	<u>Organizations</u>
D	*Europe*	*NATO, EU, UN*
TR	*Europe, Asia*	*NATO, UN*
R	*Europe, Asia*	*UN*
USA	*America*	*UN*

... expand the groups as before to 1NF ...

Join Dependencies and Multivalued Dependencies (Cont'd)

Example 7.22 (Continued)

the expanded table:

cco		
Country	**Continent**	**Organization**
D	*Europe*	*NATO*
D	*Europe*	*EU*
D	*Europe*	*UN*
TR	*Europe*	*NATO*
TR	*Europe*	*UN*
TR	*Asia*	*NATO*
TR	*Asia*	*UN*
R	*Europe*	*UN*
R	*Asia*	*UN*
USA	*America*	*UN*

There is some redundancy ...
called multivalued dependencies
cco satisfies

- *country* $\twoheadrightarrow$ *continent and*
- *country* $\twoheadrightarrow$ *organization.*

Join Dependencies and Multivalued Dependencies (Cont'd)

Example 7.22 (Continued)

cco		
Country	**Continent**	**Organization**
D	*Europe*	*NATO*
D	*Europe*	*EU*
D	*Europe*	*UN*
TR	*Europe*	*NATO*
TR	*Europe*	*UN*
TR	*Asia*	*NATO*
TR	*Asia*	*UN*
R	*Europe*	*UN*
R	*Asia*	*UN*
USA	*America*	*UN*

Actually, cco *is a join of*

encompasses	
Country	***Cont.***
D	*Europe*
TR	*Europe*
TR	*Asia*
R	*Europe*
R	*Asia*
USA	*America*

and

is_member	
Country	***Org.***
D	*EU*
D	*NATO*
D	*UN*
TR	*UN*
TR	*NATO*
R	*UN*
USA	*UN*

$$\begin{aligned} cco &= \pi[Country,Cont](cco) \bowtie \pi[Country,Org](cco) \\ &= encompasses \bowtie is\_member \end{aligned}$$

□

JOIN DEPENDENCIES (CONT'D)

Consider a set $\bar{V}$ of attributes, a relation $r \in \text{Rel}(\bar{V})$, and a decomposition $\rho = \{\bar{X}_1, \ldots, \bar{X}_n\}$ of $\bar{V}$.

r satisfies the **join dependency (JD)** $\bowtie [\bar{X}_1, \ldots, \bar{X}_n]$ if and only if

$$r = \bowtie_{i=1}^{n} \pi[\bar{X}_i](r) \ .$$

In case that $n = 2$, the JD is also called a **multivalued dependency (MVD)**, written as

$$\bar{X}_1 \cap \bar{X}_2 \twoheadrightarrow \bar{X}_1 \setminus \bar{X}_2, \text{ or, symmetrically } \bar{X}_1 \cap \bar{X}_2 \twoheadrightarrow \bar{X}_2 \setminus \bar{X}_1 \ .$$

Note: $\bar{X}_1 = (\bar{X}_1 \cap \bar{X}_2) \cup (\bar{X}_1 \setminus \bar{X}_2)$, and $\bar{X}_2 = (\bar{X}_1 \cap \bar{X}_2) \cup (\bar{X}_2 \setminus \bar{X}_1)$.

7.4.1 4. Normal Form (4NF)

Goal: mutually independent facts should not be represented in a single relation.

Consider a relation schema $R = (\bar{V}, \mathcal{D})$ where $\mathcal{D}$ is a set of MVDs and FDs. Let $\mathcal{D}^+$ the closure of $\mathcal{D}$.

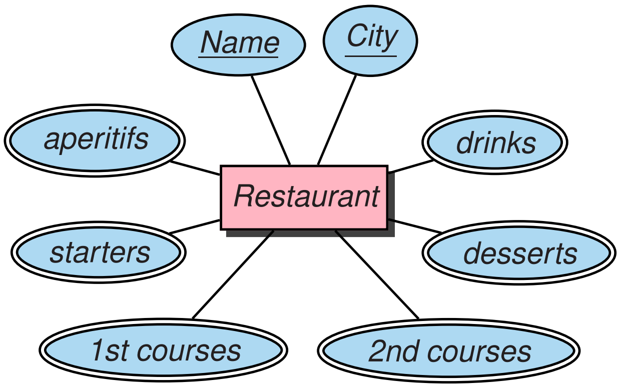
- for the closure $\mathcal{D}^+$ for MVDs see literature.
- FDs are special cases of MVDs.
- MVDs satisfy the following complement property:
  If $X \twoheadrightarrow Y \in \mathcal{D}^+$, then also $X \twoheadrightarrow (V \setminus (X \cup Y)) \in \mathcal{D}^+$.
- **trivial** MVDs are of the form $\bar{X} \twoheadrightarrow \bar{Y}$ for $\bar{Y} \subseteq \bar{X}$, and $\bar{X} \twoheadrightarrow V \setminus \bar{X}$.

Definition 7.11
A relation schema $R = (\bar{V}, \mathcal{D})$ is in 4NF if and only if for every non-trivial $\bar{X} \twoheadrightarrow Y \in \mathcal{D}^+$, $\bar{X}$ contains a key. □

Example 7.23
Consider again Example 7.22. It is not in 4NF.
Decomposition is lossless and dependency-preserving. □

Exercise 7.2

Experiment with join dependencies using the following ER diagram that describes restaurants that offer multiple choices of 2-course meals and accessoires (note that these attributes are multivalued):



7.5 Summary

- Analogous considerations for join dependencies lead to 5NF.
- 1NF $\Leftarrow$ (2NF) $\Leftarrow$ 3NF $\Leftarrow$ BCNF $\Leftarrow$ 4NF ($\Leftarrow$ 5NF)
  (other directions do not hold).
- 2NF is only of historical interest.
- In all cases there exists a lossless decomposition in 4NF (5NF).
- In the general case, all decompositions further than 3NF are not dependency-preserving.

7.6 Inclusion Dependencies

Consider sets $\bar{X}_1$ and $\bar{X}_2$ of attributes, and relations $r_1 \in \mathrm{Rel}(\bar{X}_1)$ and $r_2 \in \mathrm{Rel}(\bar{X}_2)$ with $\bar{Y} \subseteq \bar{X}_1 \cap \bar{X}_2$.

r_1, r_2 satisfy the **inclusion dependency (ID)** $R_1[\bar{Y}] \subseteq R_2[\bar{Y}]$ if and only if

$$\pi[\bar{Y}](r_1) \subseteq \pi[\bar{Y}](r_2) \; .$$

7.7 Schema Design

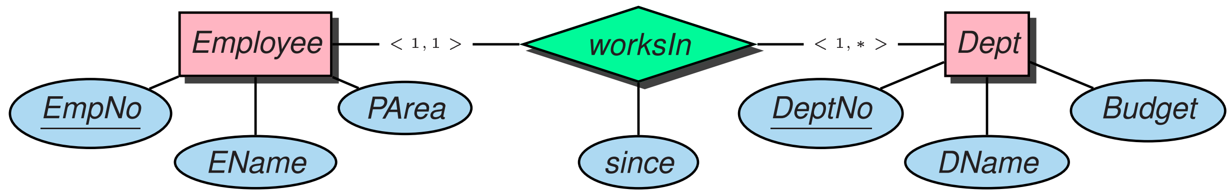
1. Generate an ER-model. This means a thorough discussion of the data engineers and the specialists of the application area.
2. Note that keys, functional dependencies, multivalued dependencies, and inclusion dependencies belong to this stage!

   Candidates can be found by data analysis, but the *semantic* aspect must be confirmed by the domain specialists.
3. Transformation to a relational schema
4. Normalization to 3NF
5. Manual decomposition to 4NF
6. enhanced ER design.

IMPORTANCE OF A CORRECT ER-DESIGN

Example 7.24
Employees are associated (uniquely) with departments. For every employee, the id, name, and the parking area must be stored. For each department, the name, the number, and the budget of the department are stored, together with the hiring date of each of the employees.

(A) An ER model:



(B) Dependency Analysis

The FD DeptNo $\rightarrow$ PArea is detected.

Inter-relational FDs are not allowed?

$\Rightarrow$ *Re-Design*

□

Chapter 8
Relational Database Languages: Relational Calculus

Overview

- Described up to now: relational algebra, SQL
- the relational calculus is a specialization of the first-order calculus, tailored to relational databases.
- straightforward: the only structuring means of relational databases are relations – each relation can be seen as an interpretation of a predicate.
- there exists a **declarative** semantics.

8.1 First-Order Logic and the Relational Calculus

The relational calculus is a specialization of first-order logic.
(This section can be skipped or compressed depending on the knowledge of the participants)

8.1.1 Syntax

- **first-order language** contains a set of distinguished symbols:
  - "(" and ")", logical symbols $\neg$, $\wedge$, $\vee$, $\rightarrow$, quantifiers $\forall$, $\exists$,
  - an infinite set of variables $X, Y, X_1, X_2, \ldots$.
- An individual first-order language is then given by its **signature** Σ. Σ contains **function symbols** and **predicate symbols**, each of them with a given arity.

For databases:

- the relation names are the predicate symbols (with arity),
  e.g. $continent/2$, $encompasses/3$, etc.
- there are only 0-ary function symbols, i.e., **constants**.
- thus, the database schema $\mathbf{R}$ is the signature.

Syntax (Cont'd).

Terms

The set of **terms** over Σ is defined inductively as

- each variable is a term,
- for every function symbol $f \in \Sigma$ with arity n and terms $t_1, \ldots, t_n$, also $f(t_1, \ldots, t_n)$ is a term.

  0-ary function symbols: c, 1,2,3,4, "Berlin",...

  Example: for $plus/2$, the following are terms: $plus(3, 4)$, $plus(plus(1, 2), 4)$, $plus(X, 2)$.
- **ground terms** are terms without variables.

For databases:

- since there are no function symbols,
- the only terms are the **constants** and **variables**
  e.g., 1, 2, "D", "Germany", X, Y, etc.

Syntax (Cont'd): Formulas

Formulas are built inductively (using the above-mentioned special symbols) as follows:

Atomic Formulas

(1) For a predicate symbol (i.e., a relation name) R of arity k, and terms $t_1, \ldots, t_k$, $R(t_1, \ldots, t_k)$ is a formula.

(2) (**for databases only, as special predicates**)
A **selection condition** is an expression of the form $t_1 \, \theta \, t_2$ where t_1, t_2 are terms, and θ is a comparison operator in $\{=, \neq, \leq, <, \geq, >\}$.
Every selection condition is a formula.

(both are also called **positive literals**)

For databases:

- the atomic formulas are the **predicates** built over relation names and these constants, e.g.,
  continent("Asia",4.5E7), encompasses("R","Asia",X), country(N,CC,Cap,Prov,Pop,A).
- comparison predicates (i.e., the "selection conditions") are atomic formulas, e.g., $X =$ "Asia", $Y > 10.000.000$ etc.

Syntax (Cont'd).

Compound Formulas

(3) For a formula F, also $\neg F$ is a formula. If F is an atom, $\neg F$ is called a **negative literal**.

(4) For a variable X and a formula F, $\forall X : F$ and $\exists F : X$ are formulas. F is called the **scope** of $\exists$ or $\forall$, respectively.

(5) For formulas F and G , the **conjunction** $F \wedge G$ and the **disjunction** $F \vee G$ are formulas.

For formulas F and G, where G (regarded as a string) is contained in F, G is a **subformula** of F.

The usual priority rules apply (allowing to omit some parentheses).

- instead of $F \vee \neg G$, the **implication** syntax $F \leftarrow G$ or $G \rightarrow F$ can be used, and
- $(F \rightarrow G) \wedge (F \leftarrow G)$ is denoted by the **equivalence** $F \leftrightarrow G$.

Syntax (Cont'd).

Bound and Free Variables

An occurrence of a variable X in a formula is

- **bound** (by a quantifier) if the occurrence is in a formula A inside $\exists X : A$ or $\forall X : A$ (i.e., in the scope of an appropriate quantifier).
- **free** otherwise, i.e.,if it is not bound by any quantifier.

Formulas without free variables are called **closed**.

Example:

- $continent($"Asia"$, X)$: X is free.
- $continent($"Asia"$, X) \wedge X > 10.000.000$: X is free.
- $\exists X : (continent($"Asia"$, X) \wedge X > 10.000.000)$: X is bound.
  The formula is closed.
- $\exists X : (continent(X, Y))$: X is bound, Y is free.
- $\forall Y : (\exists X : (continent(X, Y)))$: X and Y are bound.
  The formula is closed.

Outlook:

- closed formulas either hold in a database state, or they do not hold.
- free variables represent answers to queries:
  ?- $continent($"Asia"$, X)$ means "for which value x does $continent($"Asia"$, x)$ hold?"
  Answer: for $x = 4.5E7$.
- $\exists Y : (continent(X, Y))$: means
  "for which values x is there an y such that $continent(x, y)$ holds? – we are not interested in the value of y"
  The answer are all names of continents, i.e., that x can be "Asia", "Europe", or . . .

... so we have to **evaluate** formulas ("semantics").

8.1.2 Semantics

The semantics of first-order logic is given by **first-order structures** over the signature:

First-Order Structure

A **first-order structure** $\mathcal{S} = (I, \mathcal{D})$ over a signature Σ consists of a nonempty set $\mathcal{D}$ (**domain**) and an interpretation I of the signature symbols over $\mathcal{D}$ which maps

- every constant c to an element $I(c) \in \mathcal{D}$,
- every n-ary function symbol f to an n-ary function $I(f) : \mathcal{D}^n \to \mathcal{D}$,
- every n-ary predicate symbol p to an n-ary relation $I(p) \subseteq \mathcal{D}^n$.

For Databases:

- no function symbols with arity > 0

First-Order Structures: An Example

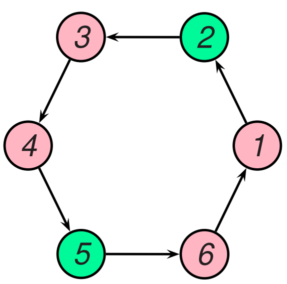
Example 8.1 (First-Order Structure)

Signature: *constant symbols:* $one,\ two,\ three,\ four,\ five,\ six$

predicate symbols: $green/1,\ red/1,\ sees/2$

function symbols: $to\_right/1,\ plus/2$

Structure $\mathcal{S}$:



Domain $\mathcal{D} = \{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$

Interpretation of the signature:

$I(one) = 1,\ I(two) = 2, \ldots, I(six) = 6$

$I(green) = \{(2),\ (5)\},\ \ I(red) = \{(1),\ (3),\ (4),\ (6)\}$

$I(sees) = \{(1,4),\ (2,5),\ (3,6),\ (4,1),\ (5,2), (6,3)\}$

$I(to\_right) = \{\ (1) \mapsto (2),\ (2) \mapsto (3),\ (3) \mapsto (4),$
$(4) \mapsto (5),\ (5) \mapsto (6),\ (6) \mapsto (1)\}$

$I(plus) = \{(n,m) \mapsto (n+m)$ *mod* $6 \mid n,m \in \mathcal{D}\}$

Terms: $one,\ to\_right(four),\ to\_right(to\_right(X)),\ to\_right(to\_right(to\_right(four))),$
$plus(X, to\_right(six)),\ to\_right(plus(to\_right(four), five))$

Atomic Formulas: $green(1),\ red(to\_right(to\_right(to\_right(four)))),\ sees(X,Y),$
$sees(X, to\_right(Z)),\ sees(to\_right(to\_right(four)), to\_right(one)),$
$plus(to\_right(to\_right(four)), to\_right(one)) = to\_right(three)$ □

Summary: Notions for Databases

- a set $\mathbf{R}$ of relational schemata; logically spoken, $\mathbf{R}$ is the **signature**,
- a database state is a structure $\mathcal{S}$ over $\mathbf{R}$
- $\mathcal{D}$ contains all domains of attributes of the relation schemata,
- for every single relation schema $R = (\bar{X})$ where $\bar{X} = \{A_1, \ldots, A_k\}$, we write $R[A_1, \ldots, A_k]$. k is the **arity** of the relation name R.
- relation names are the predicate symbols. They are interpreted by relations, e.g., $I(encompassed)$
(which we also write as $\mathcal{S}(encompassed)$).

For Databases:

- no function symbols with arity > 0
- constants are interpreted "by themselves":
$I(4) = 4$, $I($"Asia"$) =$ "Asia"
- care for domains of attributes.

Evaluation of Terms and Formulas

Terms and formulas must be **evaluated** under a given interpretation – i.e., wrt. a given database state $\mathcal{S}$.

- Terms can contain variables.
- variables are not interpreted by $\mathcal{S}$.

A **variable assignment** over a universe $\mathcal{D}$ is a mapping

$$\beta : Variables \rightarrow \mathcal{D} \; .$$

For a variable assignment β, a variable X, and $d \in \mathcal{D}$, the **modified** variable assignment β_X^d is identical with β except that it assigns d to the variable X:

$$\beta_X^d = \begin{cases} Y \mapsto \beta(Y) & \text{for } Y \neq X \; , \\ X \mapsto d & \text{otherwise.} \end{cases}$$

Example 8.2
For variables $X, Y, Z,\ \beta = \{X \mapsto 1,\ Y \mapsto \text{“Asia”}, Z \mapsto 3.14\}$ *is a variable assignment.*

$\beta_X^3 = \{X \mapsto 3,\ Y \mapsto \text{“Asia”}, Z \mapsto 3.14\}$. □

Evaluation of Terms

Terms and formulas are interpreted

- under a given interpretation $\mathcal{S}$, and
- wrt. a given variable assignment β.

For Databases:

- $\mathcal{S}$ is a database state.
- Σ is a purely relational signature,
- no function symbols with arity > 0, no nontrivial terms,
- constants are interpreted "by themselves".

Every interpretation $\mathcal{S}$ together with a variable assignment β induces an evaluation $\mathcal{S}$ of terms ($\mathcal{S}(t,\beta) \in \mathcal{D}$) and tuples of terms:

For Databases: $\mathcal{S}(x,\beta) := \beta(x)$ for a variable x ,
$\mathcal{S}(c,\beta) := c$ for a constant c .

Evaluation of Terms

Relevant only for full first-order logic:

$\mathcal{S}(x,\beta) := \beta(x)$ for a variable x ,

$\mathcal{S}(f(t_1,\ldots,t_n),\beta) := (I(f))(\mathcal{S}(t_1,\beta),\ldots,\mathcal{S}(t_n,\beta))$

for a function symbol $f \in \Sigma$ with arity n and terms $t_1,\ldots,t_n$.

Example 8.3 (Evaluation of Terms)
Consider again Example 8.1.

- *For variable-free terms:* $\beta = \emptyset$.
- $\mathcal{S}(one,\emptyset) = I(one) = 1$
- $\mathcal{S}(to\_right(four),\emptyset) = I(to\_right(\mathcal{S}(four,\emptyset)) = I(to\_right(4)) = 5$
- $\mathcal{S}(to\_right(to\_right(to\_right(four))),\emptyset) = I(to\_right(\mathcal{S}(to\_right(to\_right(four)),\emptyset))) = I(to\_right(I(to\_right(\mathcal{S}(to\_right(four),\emptyset))))) = I(to\_right(I(to\_right(5)))) = I(to\_right(6)) = 1$ □

Example 8.3 (Continued)

- *Let* $\beta = \{X \mapsto 3\}$.
  $\mathcal{S}(to\_right(to\_right(X)), \beta) = I(to\_right(\mathcal{S}(to\_right(X), \beta))) =$
  $I(to\_right(I(to\_right(\mathcal{S}(X, \beta))))) = I(to\_right(I(to\_right(\beta(X))))) =$
  $I(to\_right(I(to\_right(3)))) = I(to\_right(4)) = 5$

- *Let* $\beta = \{X \mapsto 3\}$.
  $\mathcal{S}(plus(X, to\_right(six)), \emptyset) = I(plus(\mathcal{S}(X, \beta), \mathcal{S}(to\_right(six), \beta))) =$
  $I(plus(\beta(X), I(to\_right(\mathcal{S}(six, \beta))))) = I(plus(3, I(to\_right(I(six))))) =$
  $I(plus(3, I(to\_right(6)))) = I(plus(3, 1)) = 4$ $\square$

Evaluation of Formulas

Formulas can either hold, or not hold in a database state.

Truth Value

Let F a formula, $\mathcal{S}$ an interpretation, and β a variable assignment of the free variables in F (denoted by $free(F)$).

Then we write $\mathcal{S} \models_\beta F$ if "F is true in $\mathcal{S}$ wrt. β".

Formally, $\models$ is defined inductively.

TRUTH VALUES OF FORMULAS: INDUCTIVE DEFINITION

Motivation: variable-free atoms

For an atom $R(a_1, \ldots, a_k)$, where a_i, $1 \leq i \leq k$ are constants,

$R(a_1, \ldots, a_k)$ is **true** in $\mathcal{S}$ if and only if $(I(a_1), \ldots, I(a_k)) \in \mathcal{S}(R)$.

Otherwise, $R(a_1, \ldots, a_k)$ is **false** in $\mathcal{S}$.

Base Case: Atomic Formulas

The **truth value** of an atom $R(t_1, \ldots, t_k)$, where t_i, $1 \leq i \leq k$ are terms, is given as

$$\mathcal{S} \models_\beta R(t_1, \ldots, t_k) \quad \text{if and only if } (\mathcal{S}(t_1), \ldots, \mathcal{S}(t_k)) \in \mathcal{S}(R) .$$

For Databases:

- the t_i can only be constants or variables.

TRUTH VALUES OF FORMULAS: INDUCTIVE DEFINITION

(2) $t_1 \, \theta \, t_2$ with θ a comparison operator in $\{=, \neq, \leq, <, \geq, >\}$:
$\mathcal{S} \models_\beta t_1 \, \theta \, t_2$ if and only if $\mathcal{S}(t_1, \beta) \; \theta \; \mathcal{S}(t_2, \beta)$ holds.

(3) $\mathcal{S} \models_\beta \neg G$ if and only if $\mathcal{S} \not\models_\beta G$.

(4) $\mathcal{S} \models_\beta G \wedge H$ if and only if $\mathcal{S} \models_\beta G$ and $\mathcal{S} \models_\beta H$.

(5) $\mathcal{S} \models_\beta G \vee H$ if and only if $\mathcal{S} \models_\beta G$ or $\mathcal{S} \models_\beta H$.

(6) $\mathcal{S} \models_\beta \forall X G$ if and only if for all $d \in \mathcal{D}$, $\mathcal{S} \models_{\beta_X^d} G$.

(7) $\mathcal{S} \models_\beta \exists X G$ if and only if for some $d \in \mathcal{D}$, $\mathcal{S} \models_{\beta_X^d} G$.

Example 8.4 (Evaluation of Atomic Formulas)
Consider again Example 8.1.

- *For variable-free formulas, let* $\beta = \emptyset$
- $\mathcal{S} \models_\emptyset green(1) \;\Leftrightarrow\; (1) \in I(green)$ *– which is not the case. Thus,* $\mathcal{S} \not\models_\emptyset green(1)$.
- $\mathcal{S} \models_\emptyset red(to\_right(to\_right(to\_right(four)))) \;\Leftrightarrow$
  $(\mathcal{S}(to\_right(to\_right(to\_right(four))), \emptyset)) \in I(red) \;\Leftrightarrow\; (6) \in I(red)$
  which is the case. Thus, $\mathcal{S} \models_\emptyset red(to\_right(to\_right(to\_right(four))))$.
- *Let* $\beta = \{X \mapsto 3,\ Y \mapsto 5\}$.
  $\mathcal{S} \models_\beta sees(X, Y) \;\Leftrightarrow\; (\mathcal{S}(X, \beta), \mathcal{S}(Y, \beta)) \in I(sees) \;\Leftrightarrow\; (3, 5) \in I(sees)$
  which is not the case.
- *Again,* $\beta = \{X \mapsto 3,\ Y \mapsto 5\}$.
  $\mathcal{S} \models_\beta sees(X, to\_right(Y)) \;\Leftrightarrow\; (\mathcal{S}(X, \beta), \mathcal{S}(to\_right(Y), \beta)) \in I(sees) \;\Leftrightarrow\; (3, 6) \in I(sees)$
  which is the case.
- $\mathcal{S} \models_\beta plus(to\_right(to\_right(four)), to\_right(one)) = to\_right(three) \;\Leftrightarrow$
  $\mathcal{S}(plus(to\_right(to\_right(four)), to\_right(one)), \emptyset) = \mathcal{S}(to\_right(three), \emptyset) \;\Leftrightarrow\; 2 = 4$
  which is not the case. $\square$

Example 8.5 (Evaluation of Compound Formulas)
Consider again Example 8.1.

- $\mathcal{S} \models_\emptyset \exists X : red(X) \;\Leftrightarrow$
  there is a $d \in \mathcal{D}$ *such that* $\mathcal{S} \models_{\emptyset^d_X} red(X) \;\Leftrightarrow$ *there is a* $d \in \mathcal{D}$ *s.t.* $\mathcal{S} \models_{\{X \mapsto d\}} red(X)$
  Since we have shown above that $\mathcal{S} \models_\emptyset red(6)$*, this is the case.*
- $\mathcal{S} \models_\emptyset \forall X : green(X) \;\Leftrightarrow$
  for all $d \in \mathcal{D},\ \mathcal{S} \models_{\emptyset^d_X} green(X) \;\Leftrightarrow$ *for all* $d \in \mathcal{D},\ \mathcal{S} \models_{\{X \mapsto d\}} green(X)$
  Since we have shown above that $\mathcal{S} \not\models_\emptyset green(1)$ *this is not the case.*
- $\mathcal{S} \models_\emptyset \forall X : (green(X) \vee red(X)) \;\Leftrightarrow$ *for all* $d \in \mathcal{D},\ \mathcal{S} \models_{\{X \mapsto d\}} (green(X) \vee red(X))$.
  One has now to check whether $\mathcal{S} \models_{\{X \mapsto d\}} (green(X) \vee red(X))$ *for all* $d \in domain$.
  We do it for $d = 3$:
  $\mathcal{S} \models_{\{X \mapsto 3\}} (green(X) \vee red(X)) \;\Leftrightarrow$
  $\mathcal{S} \models_{\{X \mapsto 3\}} green(X)$ *or* $\mathcal{S} \models_{\{X \mapsto 3\}} red(X) \;\Leftrightarrow$
  $(\mathcal{S}(X, \{X \mapsto 3\})) \in I(green)$ *or* $(\mathcal{S}(X, \{X \mapsto 3\})) \in I(red) \;\Leftrightarrow$
  $(3) \in I(green)$ *or* $(3) \in I(red)$
  which is the case since $(3) \in I(red)$.
- *Similarly,* $\mathcal{S} \not\models_\emptyset \forall X : (green(X) \wedge red(X))$ $\square$

8.2 Formulas as Queries

Formulas can be seen as **queries**:

- For a formula F with free variables $X_1, \ldots, X_n$, $n \geq 1$, we write $F(X_1, \ldots, X_n)$.
- each formula $F(X_1, \ldots, X_n)$ defines – dependent on a given interpretation $\mathcal{S}$ – an **answer relation** $\mathcal{S}(F(X_1, \ldots, X_n))$.

  The **answer set** to $F(X_1, \ldots, X_n)$ wrt. $\mathcal{S}$ is the set of tuples $(a_1, \ldots, a_n)$, $a_i \in \mathcal{D}$, $1 \leq i \leq n$, such that F is true in $\mathcal{S}$ when assigning each of the variables X_i to the constant a_i, $1 \leq i \leq n$.

  Formally:

  $$\mathcal{S}(F) = \{(\beta(X_1), \ldots, \beta(X_n)) \mid \mathcal{S} \models_\beta F \text{ where } \beta \text{ is a variable assignment of } free(F)\}.$$

- for $n = 0$, the answer to F is **true** if $\mathcal{S} \models_\emptyset F$ for the empty variable assignment $\emptyset$; the answer to F is **false** if $\mathcal{S} \not\models_\emptyset F$ for the empty variable assignment $\emptyset$.

Example 8.6

Consider the MONDIAL *schema.*

- *Which cities (CName, Country) have at least 1.000.000 inhabitants?*

  $$F(CN, C) = \quad \exists\, Pr, Pop, L1, L2\ (\text{city}(CN, C, Pr, Pop, L1, L2) \wedge\ Pop \geq 1000000)$$

- *Which countries (CName) belong to Europe?*

  $$\begin{aligned} F(CName) = \exists\, & CCode, Cap, Capprov, Pop, A, ContName, ContArea \\ & (\text{country}(CName, CCode, Cap, Capprov, Pop, A) \wedge \\ & \ \text{continent}(ContName, ContArea) \wedge \\ & \ ContName = \text{'Europe'} \wedge\ \text{encompasses}(ContName, CCode)\ ) \end{aligned}$$

□

Example 8.6 (Continued)

- *Again, relational division ...*
  Which organizations have at least one member on each continent

$$\begin{aligned}F(Abbrev) = \exists O, HeadqN, HeadqC, HeadqP, Est :\\ (organization(O, Abbrev, HeadqN, HeadqC, HeadqP, Est) \wedge\\ \forall Cont : ((\exists ContArea : continent(Cont, ContArea)) \rightarrow\\ \exists Country, Perc, Type : (encompasses(Country, Cont, Perc) \wedge\\ is\_member(Country, Abbrev, Type))))\end{aligned}$$

- *Negation*
  All pairs (country,organization) such that the country is a member in the organization, and all its neighbors are not.

$$\begin{aligned}F(CCode, Org) = \exists CName, Cap, Capprov, Pop, Area, Type :\\ (\textit{country}(CName, CCode, Cap, Capprov, Pop, Area) \wedge\\ \textit{is\_member}(CCode, Org, Type) \wedge\\ \forall CCode' : (\exists Length : \textit{sym\_borders}(CCode, CCode', Length) \rightarrow\\ \neg\exists Type' : \textit{is\_member}(CCode', Org, Type')))\end{aligned}$$

□

8.3 Comparison of the Algebra and the Calculus

Calculus: The semantics (= answer) of a query in the relational calculus is defined via the truth value of a formula wrt. an interpretation
“**declarative** Semantics”.

Algebra: The semantics is given by evaluating an algebraic expression (i.e., an operator tree)
“**algebraic** Semantics”.

Example: Expressing Algebra Operations in the Calculus

Consider relation schemata $R[A, B]$, $S[B, C]$, and $T[A]$.

- **Projection** $\pi[A]R$:
$$F(X) = \exists Y\, R(X, Y)$$
- **Selection** $\sigma[A = B]R$:
$$F(X, Y) = R(X, Y) \wedge X = Y$$
- **Join** $R \bowtie S$:
$$F(X, Y, Z) = R(X, Y) \wedge S(Y, Z)$$
- **Union** $R \cup (T \times \{b\})$:
$$F(X, Y) = R(X, Y) \vee (T(X) \wedge Y = b)$$
- **Difference** $R - (T \times \{b\})$:
$$F(X, Y) = R(X, Y) \wedge \neg(T(X) \wedge Y = b)$$
- **Division** $R \div T$:
$$F(Y) = \forall X : (T(X) \Rightarrow R(X, Y)) \quad \text{or} \quad F(X) = \neg\exists X : (T(X) \wedge \neg R(X, Y))$$

Safety and Domain-Independence

- If the domain $\mathcal{D}$ is infinite, the answer relations to some expressions of the calculus can be infinite!

Example 8.7
Let
$$F(X) = \neg R(X),$$
("give me all a such that $R(a)$ does not hold")

where $\mathcal{S}(R) = \{1\}$.

Depending on $\mathcal{D}$, $\mathcal{S}(F)$ is infinite. □

Example 8.8
Let
$$F(X, Z) = \exists Y (R(X, Y) \vee S(Y, Z)),$$
Consider $\mathcal{S}(R) = \{(1, 1)\}$, arbitrary $\mathcal{S}(S)$ (even empty).

Which Z? □

Example 8.9
Consider a database of persons:

married(X,Y): X is married with Y.

$F(X) = \neg married(john, X) \wedge \not(X = john)$.

What is the answer?

- *Consider* $\mathcal{D} = \{john, mary\}$, $\mathcal{S}(married) = \{(john, mary), (mary, john)\}$.
  $\mathcal{S}(F) = \emptyset$.
  - *there is no person (except John) who is not married with John*
  - *all persons are married with John???* □
- *Consider* $\mathcal{D} = \{john, mary, sue\}$, $\mathcal{S}(married) = \{(john, mary), (mary, john)\}$.
  $\mathcal{S}(F) = \{sue\}$.
  The answer depends not only on the database, but on the domain (that is a purely logical *notion)*
  Obviously, it is meant "All persons in the database who are not married with $john$*".*

Active Domain

Requirement: the answer to a query depends only on

- constants given in the query
- constants in the database

Definition 8.1
Given a formula F *of the relational calculus and a database state* $\mathcal{S}$, $DOM(F)$ *contains*

- *all constants in* F,
- *and all constants in* $\mathcal{S}(R)$ *where* R *is a relation name that occurs in* F.

$DOM(F)$ *is called the* ***active domain*** *domain of* F. □

$DOM(F)$ is finite.

Domain-Independence

Formulas in the relational calculus are required to be **domain-independent**:

Definition 8.2

A formula $F(X_1, \ldots, X_n)$ is ***domain-independent*** *if for all $D \supseteq DOM(F)$,*

$$\begin{aligned} \mathcal{S}(F) &= \{(\beta(X_1), \ldots, \beta(X_n)) \mid \mathcal{S} \models_\beta F,\ \beta(X_i) \in DOM(F) \text{ for all } 1 \le i \le n\} \\ &= \{(\beta(X_1), \ldots, \beta(X_n)) \mid \mathcal{S} \models_\beta F,\ \beta(X_i) \in D \text{ for all } 1 \le i \le n\}. \end{aligned}$$

□

It is undecidable whether a formula F is domain-independent!
(follows from Rice's Theorem).

Instead, **(syntactical) safety** is required for queries:

- stronger condition
- can be tested algorithmically

Safety

Definition 8.3

A formula F is ***(syntactically) safe*** *if and only if it satisfies the following conditions:*

1. *F does not contain $\forall$ quantifiers. (for formal simplicity since $\forall X G$ can always be replaced by $\neg\exists X \neg G$)*
2. *if $F_1 \vee F_2$ is a subformula of F, then F_1 and F_2 must have the same free variables.*
3. *for all maximal conjunctive subformulas $F_1 \wedge \ldots \wedge F_m, m \ge 1$ of F:*

   All free variables must be ***bounded****:*

   - *Let $1 \le j \le m$.*
   - *if F_j is neither a comparison, nor a negated formula, any free variable in F_j is bounded,*
   - *if F_j is of the form $X = a$ or $a = X$ with a a constant, then X is bounded,*
   - *if F_j is of the form $X = Y$ or $Y = X$ and Y is bounded, then X is also bounded.*

(a subformula G of a formula F is a ***maximal conjunctive subformula****, if there is no conjunctive subformula H of F such that G is a subformula of H).*

□

Example 8.10

- $X = Y \vee R(X, Z)$ *is not safe*
- $X = Y \wedge R(X, Y)$ *is safe*
- $R(X, Y, Z) \wedge \neg(S(X, Y) \vee T(Y, Z))$ *is not safe, but the logically equivalent formula*

$$R(X, Y, Z) \wedge \neg S(X, Y) \wedge \neg T(Y, Z)$$

is safe. □

- safety is defined purely syntactically
- safety can be tested effectively
- safety implies domain-independence
(proof by induction on the number of maximal conjunctive subformulas).

8.4 Equivalence of Algebra and (safe) Calculus

As for the algebra, the attributes of each relation are assumed to be ordered.

Theorem 8.1
For each expression Q of the relational algebra there is an equivalent safe formula F of the relational calculus, and vice versa; i.e., for every state $\mathcal{S}$, Q and F define the same answer relation. □

Proof:

(A) Algebra to Calculus

Let Q an expression of the relational algebra. The proof is done by induction over the structure of Q (as an operator tree). The formulas that are generated are always safe.

Induction base: Q does not contain operators.

- if $Q = R$ where R is a relation symbol of arity $n \geq 1$:

$$F(Z_1, \ldots, Z_n) = R(Z_1, \ldots, Z_n)$$

R	
A_1	A_2
a	b
1	2

Q: R answer to $R(Z_1, Z_2)$:

Z_1	Z_2
a	b
1	2

- otherwise, $Q = \{c\}$, $c \in \mathcal{D}$. Then, $F(Z) = (Z = c)$.

{c}
?
c

Answer to $Z = c$:

Z
c

Induction step:

Assume that Q_1 is equivalent to $F_1(X_1, \ldots, X_m)$ and Q_2 is equivalent to $F_2(Y_1, \ldots, Y_n)$.

- Case $Q = Q_1 \cup Q_2$ where $\Sigma_{Q_1} = \Sigma_{Q_2}$ and $\mid \Sigma_{Q_1} \mid = n \geq 1$.

$$\begin{array}{rclll} F(Z_1, \ldots, Z_n) & = & \exists & X_1, \ldots, \exists X_n & (F_1(X_1, \ldots, X_n) \wedge Z_1 = X_1 \wedge \ldots \wedge Z_n = X_n) \vee \\ & & \exists & Y_1, \ldots, \exists Y_n & (F_2(Y_1, \ldots, Y_n) \wedge Z_1 = Y_1 \wedge \ldots \wedge Z_n = Y_n). \end{array}$$

Example:

Q_1	
A_1	A_2
a	b
c	d

$F_1($

X_1	X_2
a	b
c	d

$)$

Q_2	
A_1	A_2
1	2
c	d

$F_2($

Y_1	Y_2
1	2
c	d

$)$

$F($

Z_1	Z_2
a	b
c	d
1	2

$)$

- Case $Q = Q_1 - Q_2$. The same, replace $\ldots \vee \ldots$ by $\ldots \wedge \neg(\ldots)$.
- Case $Q = \pi[Y]Q_1$ and $Y = \{A_{i_1}, \ldots, A_{i_k}\} \subseteq \Sigma_{Q_1}$, $k \geq 1$.

$$F(Z_1, \ldots, Z_k) = \exists X_1, \ldots, \exists X_n (F_1(X_1, \ldots, X_n) \wedge Z_1 = X_{i_1} \wedge \ldots \wedge Z_k = X_{i_k}).$$

Example:

Q_1	
A_1	A_2
a	b
c	d

$F_1($

X_1	X_2
a	b
c	d

$)$

Let $Y = \{A_2\}$: $F(Z_1) = \exists X_1, \exists X_2 (F_1(X_1, X_2) \wedge Z_1 = X_2)$

$F($

Z_1
b
d

$)$

- Case $Q = \sigma[\alpha]Q_1$, $A_i, A_j \in \Sigma_{Q_1}$ and $n \geq 1$.

$$F(X_1, \ldots, X_n) = F_1(X_1, \ldots, X_n) \wedge \alpha', \text{ where } \alpha' = \begin{cases} X_i \,\theta\, a_i & \text{for} \quad \alpha = (A_i \,\theta\, a_i), \\ a_i \,\theta\, X_i & \text{for} \quad \alpha = (a_i \,\theta\, A_i), \\ X_i \,\theta\, X_j & \text{for} \quad \alpha = (A_i \,\theta\, A_j). \end{cases}$$

Example:

Q_1	
A_1	A_2
1	2
3	4

$F_1($

X_1	X_2
1	2
3	4

$)$

Let $\sigma =$ "$A_1 = 3$": $F(Z_1, Z_2) = F_1(X_1, X_2) \wedge Z_1 = 3$

$F($

Z_1	Z_2
3	4

$)$

- Case $Q = Q_1 \bowtie Q_2$ and $\Sigma_{Q_1} = \{A_1, \ldots, A_m\}$, $\Sigma_{Q_2} = \{B_1, \ldots, B_n\}$, $n, m \geq 1$. Let w.l.o.g. $A_1 = B_1, \ldots, A_k = B_k$ for some $k \leq n, m$.

$$F(X_1, \ldots, X_m, Y_{k+1}, \ldots, Y_n) = (F_1(X_1, \ldots, X_m) \wedge F_2(Y_1, \ldots, Y_n) \wedge \\ \wedge X_1 = Y_1 \wedge \ldots \wedge X_k = Y_k).$$

Example:

Q_1	
AB_1	A_2
1	2
3	4

Q_2	
AB_1	B_2
5	6
1	7

$F_1($

X_1	X_2
1	2
3	4

$)$

$F_2($

Y_1	Y_2
5	6
1	7

$)$

$F(Z_1, Z_2, Z_3) = F_1(X_1, X_2) \wedge F_2(Y_1, Y_2) \wedge X_1 = Y_1$

$F($

Z_1	Z_2	Z_3
1	2	7

$)$

Note again that the resulting formulas F are safe.

(B) Calculus to Algebra

Consider a safe formula $F(X_1, \ldots, X_n)$, $n \geq 1$ of the relational calculus.

First, an algebra expression E that computes the active domain $DOM(F)$ of the formula and the database is derived:

Assume $R_1, \ldots, R_n$, $n \geq 0$ to be the relation names in F. For k-ary R_i,

$$E(R_i) = \pi[\$1](R_i) \cup \ldots \cup \pi[\$k](R_i).$$

Let

$$E = E(R_1) \cup \ldots E(R_n) \cup \{a_1, \ldots, a_m\},$$

where $a_j, 1 \leq j \leq m$ are the constants in F.

- $E(\mathcal{S})$ is a unary relation.

An equivalent algebra expression Q is now constructed by induction over the number of maximal conjunctive subformulas of F.

Induction base: F has exactly one maximal conjunctive subformula. Thus, $F = G_1 \wedge \ldots \wedge G_l$, $l \geq 1$.

(1) Case $l = 1$.
Then, either $F = R(a_1, \ldots, a_k)$, where a_i are variablen or constants, or F is a comparison of one of the forms $F = (X = a)$ or $F = (a = X)$, where X is a variable and a is a constante (note that all other comparisons would not be safe).

- Case $F = R(a_1, \ldots, a_k)$, e.g. $F = R(a, X, b, Y, a, X)$. Then, let

$$Q = \pi[\$2, \$4](\sigma[\Theta_1 \wedge \Theta_2](R)) \ ,$$

where

$$\Theta_1 = (\$1 = a \wedge \$3 = b \wedge \$5 = a)$$

and

$$\Theta_2 = (\$2 = \$6)$$

- Case $F = (X = a)$ or $F = (a = X)$. Let

$$Q = \{a\} \ .$$

(2) Case $l > 1$ (cf. example below) Then, w.l.o.g.

$$F = G_1 \wedge \ldots \wedge G_u \wedge G_{u+1} \wedge \ldots \wedge G_v$$

s.t. $u + v > 1$, where all G_i, $1 \leq i \leq u$ as in (1) and all G_j, $u < j \leq v$ are other comparisons.
For every G_i, $1 \leq i \leq u$ take an algebra expression $Q(G_i)$ as done in (1), where the format $\Sigma_{Q(G_i)}$ is just the set of free variables in G_i. Let

$$Q' = \bowtie_{i=1}^{u} Q(G_i).$$

With Θ the conjunction of the selection conditions $G_{u+1}, \ldots, G_v$,

$$Q = \sigma[\Theta]Q' \ .$$

Example 8.11
Consider $F = R(a, X, b, Y, a, X) \wedge S(X, Z, a) \wedge X = Y$
as $F = G_1 \wedge G_2 \wedge G_3$*:*

$$Q(G_1) = \pi[\$2, \$4](\sigma[\$1 = a \wedge \$3 = b \wedge \$5 = a \wedge \$1 = \$6](R))$$
$$Q(G_2) = \pi[\$1, \$2](\sigma[\$3 = a](S))$$

$$Q(F) = \sigma[X = Y]((([\$1 \rightarrow X, \$2 \rightarrow Y]Q(G_1)) \bowtie ([\$1 \rightarrow X, \$2 \rightarrow Z]Q(G_2)))$$

□

Induction Step: For formulas $F, G, H, \ldots$ with maximal $n-1$ maximal conjunctive subformulas, the equivalent algebra expressions are $Q(F), Q(G), Q(H), \ldots$.

(3) $F = \exists X G$.

$$Q = \pi[\$1, \ldots, \$k](Q(G)) ,$$

where G has $k+1$, $k \geq 0$ free variables, and w.l.o.g. X is the $k+1$th free variable.

(4) $F = G \vee H$.

$$Q = Q(G) \cup Q(H)$$

(safety guarantees that G and H have the same free variables, thus, $Q(G)$ and $Q(H)$ have the same format).

(5) $F = G_1 \wedge \ldots \wedge G_l$, $l \geq 1$ where some G_i are of the form $\neg H_i$. Then,

$$Q(G_i) = E^k - Q(H_i)$$

where $Q(H_i)$ is k-ary.

Q is then constructed analogous to (2).

Overview & Summary

- SQL as database language: Schema, Queries, Updates
  Transactions, Grants, Data Dictionary
  PL/SQL
  Object-Orientation, JDBC
- Database Design, Applied Database Engineering:
  ER-Model (Conceptual Model), Transformation from ER to Relational Model
  Refinement of the Relational Model (Normalization)
  Referential Integrity
- Inside the Database:
  Basic Theory: Relational Algebra
  Algebraic Optimization, Implementation
  Storage, Indexing
- Running a Database: Transactions, Safety/Recovery
- Advanced Theory: Relational Calculus