### Introduction to Databases (Winter Term 2024/2025)

Deductive Databases (Summer Term 2024)

### (c) Prof Dr. Wolfgang May Universität Göttingen, Germany

may@informatik.uni-goettingen.de

1

Introduction to Databases (BSc): 2+1/3+1 SWS, 5 ECTS: Ch. 1-3, 5; overview of 4+6 4+1 SWS: Ch. 1-6 4+2 SWS: Ch. 1-8 Database Theory/Deductive Databases (MSc): Ch. 8-12

## Chapter 1 Basic Notions

### CONTEXT AND OVERVIEW

1

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

2

### 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	declarative	
Pascal, C, C++, Java	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
Opera	ting Modes
single-user	multiuser
	<ul> <li>user accounts</li> </ul>
one-thread	concurrency
	<ul> <li>transactions</li> </ul>
	<ul> <li>safety</li> </ul>
	<ul> <li>access control</li> </ul>
	<ul> <li>against physical failure</li> </ul>
	<ul> <li>consistency, integrity</li> </ul>

4

### **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.)



### Schema Levels

**Conceptual schema:** The conceptual schema defines the model of the world as represented in the database, using an *abstract* formalism: [intended to be stable]

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

Logical schema: A mapping from the conceptual schema to a concrete data model.

**Physical (internal) schema:** Data structures for storing the data, and additional auxiliary structures for more efficient data handling (e.g., indexes). [can be changed for optimizations]

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. [easily adaptable to users' needs]

### Mappings:

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

### 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).

8

### 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.

# 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 Data Definition Language (DDL) for defining the schema: object types, relationship types, and integrity constraints.
- a Data Manipulation Language (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.

10

### DATA MODELS

Kinds of Modeling:

- conceptual modeling: abstract model of the semantics of an application
- *logical modeling*: more formal model, similar to an *abstract datatype/API* that has actual implementations

Some prominent data models:

- Network Model (1964; CODASYL Standard 1971; "legacy"); Hierarchical Model
- Entity-Relationship-Model (1976, conceptual model, only static concepts) [this lecture]
- Unified Modeling Language UML (~1995, conceptual model) [Software Engineering] comprehensive formalism for specifying processes, based on the object-oriented model.
- Relational Model (1970; simple, but clear *logical* model) [this lecture]
- XML (since 1996; popular since 1998) [Semistructured Data and XML lecture]
- RDF data model (since 1997; popular since 200X); even more basic only a single ternary relation – (subject, predicate, object) [Semantic Web]

### 2.1 Entity Relationship Model (ERM)

- purely *conceptual model*: Abstract description of the application domain in a graphical framework, which is then transformed into some logical data model (this lecture: relational model).
- This lecture uses the original "Chen Notation", named after Peter Pin-Shan Chen (born 1947 in Taichung, Taiwan; 1970-73 Harvard, 1974-78 MIT) who published it in 1976 in "The Entity-Relationship Model – Toward a Unified View of Data" in the ACM Transactions on Database Systems journal with min..max-Notation for cardinalities.
- some textbooks/lectures [e.g. the IKS lecture in our "Wirtschaftsinformatik" studies] and design tools use different notations (especially for the relationships and their cardinalities):
  - influenced by the earlier "Bachman Diagrams",
  - influenced by the later UML language (1990s);
  - most of them do not allow to model *n*-ary (n > 2) relationships directly.
  - information about the min/max-cardinalities is crucial for the mapping to the relational model.
- independent from what notation/tool you use: if you do it *correctly*, the result, i.e., the relational model obtained from the subsequent mapping step, will be the same.

12

### 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  $\{A_1, \ldots, A_n\}$ ,  $n \ge 0$  are the attributes (literal-valued properties) of a type.

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

### Example 2.1

(Continent, {name, area}) (Country, {name, code, population, area}), (City, {name, population, latitude, longitude, elevation}), (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, latitude, longitude}): {(name: Aden, population: 250000, latitude: 13, longitude: 50), (name: Kathmandu, population: 393494, latitude: 27.45, longitude: 85.25), (name: Ulan Bator, population: 479500, latitude: 48, longitude: 107) }





### **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,  $\{RO_1, \ldots, RO_k\}, k \ge 2$ , is a list of *roles*,  $\{E_1, \ldots, E_k\}$  is a list of entity types associated to the roles, and  $\{A_1, \ldots, A_n\}, n \ge 0$  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, {Country, City}, ∅), (encompasses, {Continent, Country}, {percent}), (belongsto, {Province, Country}, ∅), (flowsinto, {tributary: River, main: River}, ∅)

### **RELATIONSHIP TYPES AND RELATIONSHIP INSTANCES**

- A relationship set *b* of a relationship type *B* is a finite set of relationships.
- A relationship (instance) 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.

18

(see examples next slide)







### 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)

22

### 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 an entity amongst all entities of a given type (cf. *candidate keys, primary keys*).
- **Relationship Cardinalities:** every relationship type is assigned a cardinality 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 an entity 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 μ ∈ e, i.e., for all μ<sub>1</sub>, μ<sub>2</sub> ∈ e, if μ<sub>1</sub> and μ<sub>2</sub> have the same values for all attributes in *K*, then μ<sub>1</sub> = μ<sub>2</sub>.

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.

24

### **RELATIONSHIP CARDINALITIES**

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

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

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





26

### **Comment on Minimal Cardinalities**

- Conceptual modeling: minimal cardinality describes the allowed state of an up-and-running database:
  - 0 means the relationship is optional
  - 1 means the relationship is mandatory
- during initialization, and when new items are added, these may be temporarily violated (cf. country-capital <1,1>. How to add a new country?)

Additional Notions for Cardinalities

For binary relationships, the following notions are used:

- if max<sub>1</sub> = max<sub>2</sub> = 1, it is called a 1 : 1-relationship.
   is\_capital ⊆ Country × City is a 1:1-relationship
- if max<sub>1</sub> > 1, max<sub>2</sub> = 1, it is called a n : 1-relationship (functional relationship) from E<sub>2</sub> to E<sub>1</sub>, and a 1 : n-relationship from E<sub>1</sub> to E<sub>2</sub>.
   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  $\mu$  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).
  - such a relationship is called an *identifying relationship*,
  - the corresponding entity type is called an *identifying entity type*.
- They usually 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*).

(cases where they do not have a local key are rare, but do exist; usually resulting from *reification*, cf. Slide 38.)

• Note that weak entity types and their identifying relationship types have a special notation.

30







• The attributes and relationships of the supertype are also applicable to the subtypes (which may define further attributes and relationships).

### Generalization/Specialization

- Geographical things such as rivers, lakes, seas, mountains, deserts, and islands (no lowlands, highlands, savannas, fens, etc). All such geographical things have in common that they have names and theat they are involved in *in*-relationships with countries.
- Rivers, lakes, and seas are *waters*. These can e.g. be involved in *located-at* relationships with cities.



Generalization/Specialization

Integrity Constraints (cf. UML)

- 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.

### Intuition Annotations

- Generalization g Bottom-up: from the subclasses, the superclass is "discovered" as a general concept.
- Specialization

Top-Down: from the superclass, subclasses are "discovered" as restricted concepts.

- generalization usually leads to "covering", and in most cases also to disjointness.
- · specialization usually leads to non-covering.

### 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 can be overcome by defining artificial entity types for "the relationship".

A river flows (finally) into a sea/lake/river; more detailed, such a *relationship instance* is related to one or two countries:



36

### Aggregation

• originally introduced in J. Smith, D. Smith: Database Abstractions: Aggregation. In: Comm. of the ACM. Vol. 20, Nr. 6, 1977, pp. 405-413

Using an "aggregation entity type", this information can be specified much clearer by introducing an *aggregate type estuary* for the "river flows into another water" relationship:



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

### General Modeling Strategy: Reification

- Since the 1990s, this modeling strategy is called Reification ("turning something into a thing"), and applied in several modeling approaches (ERM, UML, XML, RDF) (UML: [Software Engineering Lecture] Association classes)
- Reification can replace ER-specific modeling concepts like *n*-ary relationship types or aggregation entity types by introducing new (usually weak) entity types, and then using *binary relationships only*.



38

### 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 (and more recently also by RDF-DBMS).

### DISCUSSION ERM (CONT'D)

- There is a more complex and more expressive language: **UML (Unified Modeling Language)**:
  - 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.

40



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 cardinalities (0,1) and (1,\*). Investigate all possible ways how to assign these relationship cardinalities 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.

### 42

### 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 cardinalities?
- 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.

### **DEVELOPMENT OF A DATABASE APPLICATION**

(cf. 3-Level-Architecture, Slide 6)

- **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).

... but this cannot be "used" in a real database.

**Implementation Design:** convert into the actual, *logical* schema of the logical level in a logical model (Relational Model),

44

(process to be continued then on Slide 49)

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
- "First Normal Form": only domains of atomic datatypes, no records, lists, sets etc.
- A (relational) database schema R is given by a (finite) set of (relation) schemata. Continent: ...; Country: ...; City: ...; encompasses: ...; isMember: ...
- for every relation, a set of (primary) key attributes is distinguished

### 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	10523000		
Africa	30221500		
Asia	44614500		
North America	24709000		
South America	17840000		
Australia	9000000		

46

### Relations: Example

Contine	ent	Country				
<u>name</u>	area	name	code	population	area	
Europe	10523000	Germany	D	83536115	356910	
Africa	30221500	Sweden	S	8900954	449964	
Asia	44614500	Russia	R	143666931	17075200	
North America	24709000	Poland	PL	38642565	312683	
South America	17840000	Bolivia	BOL	1098580	7165257	
Australia	\$000000					

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



### **DEVELOPMENT OF A DATABASE APPLICATION**

(cf. 3-Level-Architecture, Slide 6 and Slide 44)

**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).

- · this mapping is described next,
- then realize it in the database (SQL) ...

### DEVELOPMENT OF A DATABASE APPLICATION (CONT'D)

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

 don't worry: creating the logical schema in an SQL database *automatically* creates a structure on the physical level (this is the advantage of having the relational model as a kind of an abstract datatype that is implemented in a standardized way by relational databases).

Detailed Physical Design: optionally/later: finetuning of the physical level.

### Implementation of the External Level:

- clarify the requirements on the external level by using the conceptual model, adapt to daily users' needs (forms, presentations, reports, data exchange interfaces, ...),
- implement the external level based on the logical model.

Note:

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

50

### 2.3 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  $R_{ER}$  a relationship type in the ERM.

- Entity types:  $(E_{ER}, \{A_1, ..., A_n\}) \to E(A_1, ..., A_n),$
- For weak entity types, the key attributes of the identifying entity type must be added.
- Relationship types:

 $\begin{array}{l} (R_{ER}, \{RO_1 : E_1, \ldots, RO_k : E_k\}, \{A_1, \ldots, A_m\}) \to \\ B(E_1\_K_{11}, \ldots, E_1\_K_{1p_1}, \ldots, E_k\_K_{k1}, \ldots, E_k\_K_{kp_k}, A_1, \ldots, A_m) , \\ \text{where } \{K_{i1}, \ldots, K_{ip_i}\} \text{ are the primary keys of } E_i, 1 \leq i \leq k. \end{array}$ 

- 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 cardinality, 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**

 $(E_{ER}, \{A_1, \ldots, A_n\}) \to E(A_{i_1}, \ldots, A_{i_k})$ 

where  $\{A_{i_1}, \ldots, A_{i_k}\} \subseteq \{A_1, \ldots, A_n\}$  are the scalar (i.e., not multivalued) attributes of  $E_{ER}$  – multivalued attributes are mapped separately.



Continent				
name	area			
VARCHAR(20)	NUMBER			
Europe	10523000			
Africa	30221500			
Asia	44614500			
North America	24709000			
South America	17840000			
Australia	9000000			

The candidate keys of the relation are the candidate keys of the entity type.

52

### MULTIVALUED ATTRIBUTES

... one thing left:

Attributes of relations must only be single values.

 $(E_{ER}, \{A_1, \dots, A_i, \dots, A_n\})$  where  $A_i$  is a multivalued attribute  $\rightarrow E\_A_i(K_1, \dots, K_p, A_i)$ 

where  $\{K_1, \ldots, K_p\}$  are the primary keys of *E*.

(renaming is allowed, especially if there is only one key attribute)

 $\{K_1, \ldots, K_p, A_i\}$  are the primary keys of the relation  $E\_A_i$ .



Languages			
country	language		
D	German		
СН	German		
СН	French		

### 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	
Freiburg	СН	FR	NULL	
Cordoba	E	Andalucia	328326	
Cordoba	RA	Cordoba	1207774	
••				

54

### **RELATIONSHIP TYPES**

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



• Note: for references to weak entity types, the global key must be used (exercise: located\_on as an *n*:*m* relationship between cities and islands).

Relationship Types: 1:n-Relationships

In case that k = 2 (binary relationship) and a (0,1)- or (1,1)-relationship cardinality (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)

Example: the "capital" relationship can be merged into the "country" table (or, less intuitively, into the "city" table, where the <0,1> indicates that it would generate lots of NULL values).

name)-	Country – <u>code</u>		Country				
Germany			Country			1	1
Germany	< 1, 1 >	name	<u>code</u>	population	capital	province	
		Germany	D	83536115	Berlin	Berlin	
$\langle$	is_capital	Austria	А	8023244	Vienna	Vienna	
		Canada	CDN	28820671	Ottawa	Quebec	
	< 0, 1 >	Bolivia	BOL	7165257	La Paz	Bolivia	
name -	City pop.						
Berlin	3472009	)					

Other examples: headquarters of organizations, flows into (the latter is a bit more complex because a river flows into another river, a lake, or a sea).

56

### **Recursive Symmetric Relationship Types**

- recursive non-symmetric relationship types (river-flowsInto): use role names as column names.
- recursive symmetric relationship types (borders): invent column names
  - Symmetric storage would introduce redundancy and risk for inconsistencies. Store only one direction and create a symmetric SQL view from it (belongs to the "external level" of the 3-level-architecture)



borders				
country1	country2	length		
D	F	451		
F	Ð	450		
D	СН	334		
СН	F	573		
:	:	:		

SQL view (see later)

CREATE VIEW symmborders AS (SELECT \* FROM borders) UNION (SELECT country2, country1, length FROM borders)

### EXERCISE

### Exercise 2.4

Give a relational schema for the following ER schema:



### 2.4 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  $\bar{X} = \{A_1, \dots, A_m\}$ ,  $m \ge 1$  of attributes.

 $\bar{X}$  is the **format** of the schema.

- a (relational) database schema 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 attribute A ∈ X
   is associated to a (non-empty) domain of atomic values, called dom(A).
- $\operatorname{dom}(\bar{X}) := \operatorname{dom}(A_1) \times \ldots \times \operatorname{dom}(A_m).$
- Example: Continent(name, area) dom(continent.name) = VARCHAR, dom(continent.area) = NUMBER dom(Continent) = VARCHAR × NUMBER

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.

60

### **RELATIONAL DATABASES – FORMALIZATION: SEMANTICS**

- A (relational) database (or, more explicitly, a database state) S (over  $\mathbf{R} = \{R_1(\bar{X}_1), \dots, R_n(\bar{X}_n)\}$ ) is a relational structure over  $\mathbf{R}$ .
- A relational structure S associates each  $R_i(\bar{X}_i)$  to a relation  $S(R_i)$  over  $\bar{X}_i$ .
- elements of a relation are called tuples.
   (every tuple represents an entity or a relationship.)
   Note: as a set of tuples, there cannot be two tuples that have all the same values.
- a **tuple**  $\mu$  over  $\bar{X}$  is a mapping  $\mu : \bar{X} \to \operatorname{dom}(\bar{X})$ ; or, for each individual attribute,  $\mu : A \to \operatorname{dom}(A)$ .

 $\operatorname{Tup}(\bar{X})$  denotes the set of all tuples over  $\bar{X}$ .

Example: Consider tuples in the Continent(name, area) table:  $\mu =$  name  $\mapsto$  "Asia", area  $\mapsto$  44614500 with  $\mu$ (name) = "Asia",  $\mu$ (area) = 44614500

- a relation r over  $\overline{X}$  is a finite set  $r \subseteq \text{Tup}(\overline{X})$  usually represented by a table.
- $\operatorname{Rel}(\bar{X}) := 2^{Tup(\bar{X})}$  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)

62

### Keys

While in the ER model, the keys serve only for an *intuitive* modeling, in relational database design they play an important role for the database performance and for the ability of the database to incorporate and maintain *key constraints*.

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

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

If  $\mu_1(\bar{K}) = \mu_2(\bar{K})$  (i.e.,  $\mu_1$  and  $\mu_2$  coincide on the values of  $\bar{K}$ ), then  $\mu_1 = \mu_2$ .

More Concrete Requirements on Keys

(to be formalized on the next slides)

- keys should be minimal: no subset  $\bar{K}' \subsetneq \bar{K}$  satisfies the key property,
- no "embedded relations" (i.e., partial functional dependencies): for no subset X
  <sup>'</sup> ⊊ X
  <sup>'</sup> of the attributes of R, any subset K
  <sup>'</sup> ⊊ K
  <sup>'</sup> satisfies the key property wrt. X
  <sup>'</sup>.
  [3rd Normal Form, cf. Slide 374; Example see Slide 67]

### KEYS: ADDITIONAL FORMAL REQUIREMENTS

The relational model provides a more concise formalization of keys (cf. Slide 326 ff. on Normalization Theory for details).

These are based on the definition of functional dependencies:

Given a relation  $R(\bar{X})$ ,  $\bar{V}, \bar{W} \subseteq \bar{X}$ . r satisfies the **functional dependency (FD)**  $\bar{V} \to \bar{W}$  if for all tuples  $\mu_1, \mu_2 \in r$ ,

 $\mu_1(\bar{V}) = \mu_2(\bar{V}) \Rightarrow \mu_1(\bar{W}) = \mu_2(\bar{W}) .$ 

(" $\bar{W}$  functionally depends on  $\bar{V}$ ")

### Example 2.4

Consider the relation schema Country(name, code, area, population, capital, capprov).

The following functional dependencies hold wrt. the intended application domain:

 $\begin{array}{ll} \{ code \} \rightarrow \{ name \}, & \{ name \} \rightarrow \{ code \} \\ \{ code \} \rightarrow \{ area, \ population, \ capital, \ capprov \} \\ \{ code \} \rightarrow \{ name, \ code, \ area, \ population, \ capital, \ capprov \} \\ \{ name \} \rightarrow \{ name, \ code, \ area, \ population, \ capital, \ capprov \} \end{array}$ 

64

### Keys (Cont'd)

• 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.

### KEYS: ADDITIONAL FORMAL REQUIREMENTS

• Formalization of the **Key Constraint**:  $\bar{K} \subseteq \bar{X}$  is a possible key of  $R(\bar{X})$  if  $\bar{K} \to \bar{X}$ .

### Additionally:

- every single attribute should be *fully dependent* on the *complete* key: for every A ∈ (X̄ \ K̄): there is no subset K̄' ⊊ K̄ s.t. K̄' → A.
  (otherwise: if there is some attribute that depends only on a part of the key, split this relationship into a separate table, cf. example on Slide 67 and section on Normalization Theory, Slide 326.)

Although looking formally, the second criterion is also easy to understand and prevents bad/dangerous database design.

66	
00	

Keys and Database Design: Example								
Country (bad schema)								
Name	Code	Language	Percent	Population	Area	Capital	Province	
Germany	D	German	100	83536115	356910	Berlin	Berlin	
Switzerland	СН	German	65	7207060	41290	Bern	BE	
Switzerland	СН	French	18	7207060	41290	Bern	BE	
Switzerland	СН	Italian	12	7207060	41290	Bern	BE	
:	:	:	:	:	:	:	:	

- the database is redundant
- needs more space, less efficient to query
- update anomalies/risks: updating Swiss population requires to update all three lines, otherwise inconsistent information

Dependency analysis:

Keys: {Code, Language} or {Name, Language}, but e.g. already {Code}  $\rightarrow$  {Population, Capital}

Split into Country(Name, <u>Code</u>, Population, Capital, Province) and Languages(<u>Code</u>, <u>Language</u>, Percent).

### Keys and Database Design

- A good ER model and straightforward translation as introduced in the previous section leads to a good relational design
- determining the keys is helpful in validating the design:
- for tables obtained from translating entity types, the keys are the same as in the ER model (for weak entity types: including those of the identifying entity types; cf. Country)
- the handling of multivalued attributes as shown on Slide 53 is a consequence of the functional dependency analysis (same case as in the above example)
- for relations that represent relationship types: see exercise below.

### Exercise: Keys of relations obtained from relationships

Discuss how the keys of the relations that are obtained from relationship types are determined. Which alternative scenarios have to be considered?

- · consider binary relationships systematically,
- · what about ternary relationships?

68

### INCLUSION CONSTRAINTS AND REFERENTIAL INTEGRITY

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

 $r_1 = S(R_1)$  and  $r_2 = S(R_2)$  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  $R_2.\bar{Y}_2$ .
  - $\bar{Y}_1$  is called a **foreign key** in  $R_1$  that references  $R_2.\bar{Y}_2$ .
- encompasses.continent  $\subseteq$  Continent.name
- encompasses.country ⊆ Country.code

Referential integrity constraints result from incorporating the *keys* of the participating entities into the table that represents the relationship.

### 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" (e.g. "last eruption date" for mountains other than volcanoes)
- a **partial tuple** over  $\bar{X}$  is a mapping s.t.

for all  $A \in \overline{X}$ ,  $\mu(A) \in \mathsf{dom}(A) \cup \{null\}$ .

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

70

### 2.4.1 Exercise

### Exercise 2.5

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

- Give  $\operatorname{Tup}(\bar{X})$  and  $\operatorname{Rel}(\bar{X})$ .
- A is a key of R. Which relations  $r \in \operatorname{Rel}(\bar{X})$  violate the key constraint?
# Chapter 3 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.

72

# **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.

# 3.1 Relational Algebra: Computations over Relations

# Operations on Tuples – Overview Slide

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

(Formal definition of  $\mu$  see Slide 61)

- For Ø ⊂ Ī ⊆ X̄, the expression μ[Ī] denotes the projection of μ to Ī.
   Result: μ[Ī] ∈ Tup(Ī) where μ[Ī](A) = μ(A), A ∈ Ī.
- A selection condition α (wrt. X̄) is an expression of the form A θ B or A θ c, or c θ A where A, B ∈ X̄, dom(A) = dom(B), c ∈ dom(A), and θ is a comparison operator on that domain like e.g. {=,≠,≤,<,≥,>}.

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

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

• For  $\overline{Y} = \{B_1, \dots, B_k\}$ , the expression  $\mu[A_1 \to B_1, \dots, A_k \to B_k]$  denotes the **renaming** of  $\mu$ .

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

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

Projection (Reduction to a subset of the attributes)

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

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

# projection to a given set of attributes

# Example 3.1

Consider the relation schema  $R(\bar{X}) = Continent(name, area): \bar{X} = [name, area]$ and the tuple  $\mu = \boxed{name \mapsto \text{``Asia'', area} \mapsto 4.50953e+07}$ . formally:  $\mu(name) = \text{``Asia'', } \mu(area) = 4.5E7$ projection attributes: Let  $\bar{Y} = [name]$ Result:  $\mu[name] = \boxed{name \mapsto \text{``Asia''}}$ 

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

Selection (only those tuples that satisfy some condition)

A selection condition  $\alpha$  (wrt.  $\bar{X}$ ) is an expression of the form  $A \theta B$  or  $A \theta c$ , or  $c \theta A$  where  $A, B \in \bar{X}$ , dom $(A) = \text{dom}(B), c \in \text{dom}(A)$ , and  $\theta$  is a comparison operator on that domain like e.g.  $\{=,\neq,\leq,<,\geq,>\}$ .

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

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

# Example 3.2

Consider again the relation schema  $R(\bar{X}) = continent(name, area)$ :  $\bar{X} = [name, area]$ .

Selection condition: *area* > 20000000.

Consider again the tuple  $\mu = |$  name  $\mapsto$  "Asia", area  $\mapsto$  4.50953e+07

formally:  $\mu(name) =$  "Asia",  $\mu(area) = 4.5E7$ 

*check:*  $\mu(area) > 20000000$ 

Result: yes.

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

76

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

Renaming (of attributes)

For  $\overline{Y} = \{B_1, \dots, B_k\}$ , the expression  $\mu[A_1 \to B_1, \dots, A_k \to B_k]$  denotes the **renaming** of  $\mu$ . Result:  $\mu[\dots, A_i \to B_i, \dots] \in \text{Tup}(\overline{Y})$  where  $\mu[\dots, A_i \to B_i, \dots](B_i) = \mu(A_i)$  for  $1 \le i \le k$ .

# renaming of attributes (without changing the tuple)

# Example 3.3

Consider (for a tuple of the table  $R(\bar{X}) = encompasses(country, continent, percent)$ ):  $\bar{X} = [country, continent, percent].$ Consider the tuple  $\mu = \boxed{country \mapsto "R", continent \mapsto "Asia", percent \mapsto 80}$ . formally:  $\mu(country) = "R", \mu(continent) = "Asia", \mu(percent) = 80$ Renaming:  $\bar{Y} = [code, name, percent]$ Result: a new tuple  $\mu[country \rightarrow code, continent \rightarrow name, percent \rightarrow percent] = <math>\boxed{code \mapsto "R", name \mapsto "Asia", percent \mapsto 80}$  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.

78

# INDUCTIVE DEFINITION OF EXPRESSIONS

Atomic Expressions - Base Cases of the Inductive Definition

 For an arbitrary attribute A and a constant c ∈ dom(A), the constant relation A : {c} is an algebra expression.

Format: [A] Result relation:  $\{\mu\}$  with  $\mu = (A \mapsto c)$ 

A:{c}	
Α	
С	

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

Format of  $R_i$ :  $X_i$ 

Result relation (wrt. a given database state S): the relation  $S(R_i)$  that is currently stored in the database.



80

# **BASE OPERATORS**

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

#### Union

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

	A	В	C			Б	~		A	В	C
-		-	-		A	В	C		a	b	c
r =	a	b G	C f	s =	b	g	a	$r \cup s =$	d	a	f
	u c	$\begin{array}{c} a \\ b \end{array}$	J d		d	a	f		c	b	d
	U	0	u						b	g	a

(note: no duplicates in the result - based on set theory)

#### Set Difference

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

$$r = \frac{A \quad B \quad C}{a \quad b \quad c} \qquad s = \frac{A \quad B \quad C}{b \quad g \quad a} \qquad r \setminus s = \frac{A \quad B \quad C}{a \quad b \quad c} \\ \frac{A \quad B \quad C}{c \quad b \quad d} \qquad r \setminus s = \frac{A \quad B \quad C}{c \quad b \quad d}$$

82

# Projection (Reduction to a subset of the attributes)

Assume  $r \in \operatorname{Rel}(\bar{X})$  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 3.4

Continent						
<u>name</u>	area					
Europe	10523000					
Africa	30221500					
Asia	44614500					
N. America	24709000					
S. America	17840000					
Australia	9000000					

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

$\mu_1[name] =$	$\mathit{name}\mapsto \mathit{``Europe''}$
$\mu_2[name] =$	$\mathit{name}\mapsto \mathit{``Africa"}$
$\mu_3[name] =$	$\mathit{name}\mapsto \mathit{``Asia"}$
$\mu_4[name] =$	name → "N.America"
$\mu_4[name] =$	name → "S.America"
$\mu_5[name] =$	$\mathit{name}\mapsto \mathit{``Australia''}$

$\pi[name]$ ( <b>Continent</b> )	
name	
Europe	
Africa	
Asia	
N.America	
S.America	
Australia	Г

# Selection (Reduction of number of tuples by a condition)

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

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

### Example 3.5

Continent		Let $\alpha = "area > 20000000"$			
<u>name</u>	area		$\sigma$ [area > 20E6]( <b>Contin</b>		
Europe	10523000	$\mu_1(area) > 20000000$ ?- no			
Africa	30221500	$\mu_2(area) > 20000000? - ves$	name	area	
Acia	44614500	$\mu_2(arror) > 200000002$	Africa	30221500	
ASIA	44014500	$\mu_3(area) > 20000000? - yes$	Asia	44614500	
N. America	24709000	$\mu_4(area) > 20000000$ ?- yes	710/4	11011000	
S. America	17840000	$\mu_4(area) > 20000000?$ — no	N.America	24709000	
Australia	9000000	$\mu_5(area) > 20000000$ ?- no			

84

# Renaming (of attributes)

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

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

#### Example 3.6

Consider the renaming of the table encompasses(country, continent, percent):

 $\bar{X} = [country, continent, percent]$ Renaming:  $\bar{Y} = [code, name, percent]$ 

$\rho[coun$	$\rho[country \rightarrow code, \ continent \rightarrow name, \ percent \rightarrow percent](encompasses)$					
code	name	percent				
R	Europe	20				
R	Asia	80				
D	Europe	100				
:	:	••••				

ssume $r\in {\sf Rel}(ar X)$ and $s\in$	$Rel(\bar{Y})$ for arbitrary	$\bar{X}, \bar{Y}.$				
convention: For $ar{X}\cupar{Y}$ , as for two tuples $\mu_1=v_1,\dots$	a shorthand, write $\overline{X}$ , $v_n$ and $\mu_2 = w_1$	$\overline{Y}$ .	:=[	$v_1,.$	$\ldots, v$	$w_n, w_1, \ldots, w_n$
tesult format of $r \bowtie s$ : $\overline{XY}$ tesult relation: $r \bowtie s = \{\mu$	$\in \operatorname{Tup}(\overline{XY}) \mid \mu[\bar{X}] \in \mathbb{R}$	$r  ext{ and } \mu[ar{Y}] \in s \}$				
Motivation						
Simplest Case: $\bar{X} \cap \bar{V} = \emptyset$		ot a balance a balance				
$r \times s = \{\mu_1 \mu_2 \in Tup(\overline{XY})\}$	$  \mu_1 \in r \text{ and } \mu_2 \in s \}.$	$\operatorname{CL} r \bowtie s = r \times s$				
$r \times s = \{\mu_1 \mu_2 \in Tup(\overline{XY})\}$	$\Rightarrow$ Cartesian Production $\mu_1 \in r$ and $\mu_2 \in s$ .	$\operatorname{ct} r \bowtie s = r \times s$	A	В	C	D
$r \times s = \{\mu_1 \mu_2 \in Tup(\overline{XY})\}$	$\Rightarrow$ Cartesian Produc $  \mu_1 \in r \text{ and } \mu_2 \in s \}.$	$\operatorname{ct} r \bowtie s = r \times s$	A 1	<i>B</i> 2	C a	$\frac{D}{b}$
$r \times s = \{\mu_1 \mu_2 \in Tup(\overline{XY})$	$\Rightarrow$ Cartesian Product $  \mu_1 \in r \text{ and } \mu_2 \in s \}.$ $\frac{C  D}{a  b}$	$\operatorname{Cl} r \bowtie s = r \times s$	$\frac{A}{1}$	B 2 2	$C = a \\ c$	$\begin{array}{c} D \\ b \\ d \end{array}$
$r \times s = \{\mu_1 \mu_2 \in Tup(\overline{XY})$ $r = \frac{A - B}{1 - 2}$	$  \mu_1 \in r \text{ and } \mu_2 \in s \}.$ $s = \frac{C  D}{a  b}$	$r \bowtie s = r \times s$	A 1 1 1	B 2 2 2	C a c e	$egin{array}{c} D \ b \ d \ f \end{array}$
$r \times s = \{\mu_1 \mu_2 \in Tup(\overline{XY})$ $r = \frac{A  B}{1  2}$ $4  5$	$  \mu_{1} \in r \text{ and } \mu_{2} \in s \}.$ $s = \frac{C  D}{a  b}$ $c  d$	$r \bowtie s = r \times s$	$\begin{array}{c} A \\ \hline 1 \\ 1 \\ 1 \\ 4 \end{array}$	B 2 2 2 5	C a c e a	$\begin{array}{c} D\\ b\\ d\\ f\\ b \end{array}$
$r \times s = \{\mu_1 \mu_2 \in Tup(\overline{XY})$ $r = \frac{A  B}{1  2}$ $4  5$	$  \mu_{1} \in r \text{ and } \mu_{2} \in s \}.$ $s = \frac{C  D}{a  b}$ $c  d$ $e  f$	$r \bowtie s = r \times s$	$\begin{array}{c} A \\ 1 \\ 1 \\ 1 \\ 4 \\ 4 \end{array}$	B 2 2 2 5 5	C a c e a c	$\begin{array}{c} D\\ b\\ d\\ f\\ b\\ d \end{array}$

86

# Example 3.7 (Cartesian Product of Continent and Encompasses)

The cartesian product combines everything with everything, not only "meaningful" combinations:

Continent × encompasses									
name	area	continent	country	percent					
Europe	10523000	Europe	D	100					
Europe	10523000	Europe	R	20					
Europe	10523000	Asia	R	80					
Europe	10523000	:	:	:					
Africa	30221500	Europe	D	100					
Africa	30221500	Europe	R	20					
Africa	30221500	Asia	R	80					
Africa	30221500	:	:	:					
Asia	44614500	Europe	D	100					
Asia	44614500	Europe	R	20					
Asia	44614500	Asia	R	80					
Asia	44614500	:	:	:					
:	:	:	:	:					

#### Back to the Natural Join

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

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

(Note: this implies that the tuples  $\mu_1 := \mu[\bar{X}] \in r$  and  $\mu_2 := \mu[\bar{Y}] \in s$  coincide in the shared attributes  $\bar{X} \cap \bar{Y}$ )

# Example 3.8

Consider encompasses(country,continent,percent) and isMember(organization,country,type):

encompasses				is	Member	
country	continent	percent		organization	country	type
R	Europe	20		EU	D	member
R	Asia	80		UN	D	member
D	Europe	100		UN	R	member
:	:	:		:	:	:

 $encompasses \bowtie isMember = \{\mu \in Tup(country, cont, perc, org, type) \mid$ 

 $\mu[country, cont, perc] \in encompasses \text{ and } \mu[org, country, type] \in isMember\}$ 

88

**Example 3.8 (Continued)**  $encompasses \bowtie isMember = \{\mu \in Tup(country, cont, perc, org, type) \mid$  $\mu[country, cont, perc] \in encompasses and \mu[org, country, type] \in isMember\}$ start with  $(R, Europe, 20) \in encompasses$ . check which tuples in *isMember* match:  $(UN, R, member) \in isMember matches:$ result: (R, Europe, 20, UN, member) belongs to the result. (some more matches ...) continue with  $(R, Asia, 80) \in encompasses$ .  $(UN, R, member) \in isMember matches:$ result: (R, Asia, 80, UN, member) belongs to the result. (some more matches ...) continue with  $(D, Europe, 100) \in encompasses.$  $(EU, D, member) \in isMember matches:$ result: (D, Europe, 100, EU, member) belongs to the result.  $(UN, D, member) \in isMember matches:$ result: (D, Europe, 100, UN, member) belongs to the result. (some more matches ...) 

# **Example 3.8 (Continued)**

Result:

encompasses 🖂 isMember									
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	:	:					
:	:	:	:	:					

#### 90

# Example 3.9 (and Exercise)

Consider the expression

 $Continent \bowtie \rho[country \rightarrow code, \ continent \rightarrow name, \ percent \rightarrow percent](encompasses) \qquad \Box$ 

Functionalities of the Join

- · Combining relations
- Selective functionality: only matching tuples survive (consider joining cities and organizations on headquarters)

# **DERIVED OPERATORS**

#### Intersection

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

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

# **Theorem 3.1** Intersection can be expressed by difference: $r \cap s = r \setminus (r \setminus s)$ .

#### $\theta ext{-Join}$

Combination of Cartesian Product and Selection:

Assume  $r \in \text{Rel}(\bar{X})$ , and  $s \in \text{Rel}(\bar{Y})$ , such that  $\bar{X} \cap \bar{Y} = \emptyset$ , and  $A \theta B$  a selection condition.  $r \bowtie_{A\theta B} s = \{\mu \in \text{Tup}(\overline{XY}) \mid \mu[\bar{X}] \in r, \ \mu[\bar{Y}] \in s \text{ and } \mu \text{ satisfies } A\theta B\} = \sigma[A\theta B](r \times s).$ 

 $I \bowtie_{A \theta B} s = \{ \mu \in \mathsf{rop}(XI) \mid \mu[X] \in I, \ \mu[I] \in S \text{ and}$ 

Equi-Join

 $\theta$ -join that uses the "="-predicate.

# Example 3.10 (and Exercise)

Consider again Example 3.7:

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

 $Continent \bowtie_{continent=name} encompasses$  would be more useful.

# Furthermore, consider

 $\pi$ [continent, area, code, percent](Continent  $\bowtie_{continent=name}$  encompasses):

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

#### 92

#### Semi-Join

- · recall: joins combine, but are also selective
- semi-join acts like a selection on a relation r: selection condition not given as a boolean formula on the attributes of r, but by "looking into" another relation (a subquery)

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

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

The semi-join  $r \bowtie s$  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"):

- Used with subqueries: (main query)  $\bowtie$  (subquery)
- $r \bowtie s \subseteq r$
- Used for optimizing the evaluation of joins (often in combination with indexes).



# Towards the Outer Join

• The (inner) join is the operator for combining relations

#### Example 3.11

• Persons work in divisions of a company, tools are assigned to the divisions:

PersonDivisionDivisionToolPersonDivisionJohnProductionProductionhammerJohnProductionBillProductionResearchpenBillProduction	Works 🖂 Tools			
JohnProductionProductionhammerJohnProductionBillProductionResearchpenBillProduction	sion Tool	1		
Bill Production Research pen Bill Production	luction hamm	nmer		
	luction hamm	nmer		
John Research Research computer John Res	earch pen			
Mary Research Admin. typewriter John Res	earch compl	nputer		
Sue Sales Mary Res	earch pen			

- join contains no tuple that describes Sue,
- join contains no tuple that describes the administration or sales division,
- join contains no tuple that shows that there is a typewriter.

computer

Mary

Research

# Outer Join

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

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

The outer join extends the "inner" join with all tuples that have no counterpart in the other relation (filled with null values):

# Example 3.12 (Outer Join)

Consider again Example 3.11

Works in Tools		
Person	Division	ΤοοΙ
John	Production	hammer
Bill	Production	hammer
John	Research	pen
John	Research	computer
Mary	Research	pen
Mary	Research	computer
Sue	Sales	NULL
NULL	Admin	typewriter

Works 🖂 Tools		
Person	Division	
John	Production	
Bill	Production	
John	Research	
Mary	Research	

Works >>> Tools		
Division	ΤοοΙ	
Production	hammer	
Research	pen	
Research	computer	

96

Formally, the result relation  $r \equiv s$  is defined as follows:  $J = r \bowtie s$  — take the ("inner") join as base  $r_0 = r \setminus \pi[\bar{X}](J) = r \setminus (r \bowtie s) - r$ -tuples that "are missing"  $s_0 = s \setminus \pi[\bar{Y}](J) = s \setminus (r \Join s) - s$ -tuples that "are missing"  $ar{Y}_0 = ar{Y} \setminus ar{X}, \, ar{X}_0 = ar{X} \setminus ar{Y}$ Let  $\mu_s \in \text{Tup}(\bar{Y}_0)$ ,  $\mu_r \in \text{Tup}(\bar{X}_0)$  such that  $\mu_s, \mu_r$  consist only of *null* values  $r \sqsupset s = J \cup (r_0 \times \{\mu_s\}) \cup (s_0 \times \{\mu_r\}).$ Example 3.12 (Continued) For the above example,  $J = Works \bowtie Tools$  $r_0 = [$ "Sue", "Sales" $], s_0 = [$ "Admin", "Typewriter"] $\bar{Y}_0 =$  Tool,  $\bar{X}_0 =$  Person Tool Person  $\mu_s =$  $\mu_r =$ null null **Division** Tool **Division** Tool Person Person  $s_0 \times \{\mu_r\} =$  $r_0 \times \{\mu_s\} =$ Sue Sales null null Admin Typewriter

# Left and Right Outer Join

# Analogously to the (full) outer join:

- $r \supset s = J \cup (r_0 \times \{\mu_s\})$ .
- $r \bowtie s = J \cup (s_0 \times \{\mu_r\})$ .

# Generalized Natural Join

Assume  $r_i \subseteq \operatorname{Tup}(\bar{X}_i)$ . Result format:  $\bigcup_{i=1}^n \bar{X}_i$ Result relation:  $\bowtie_{i=1}^n r_i = \{\mu \in \operatorname{Tup}(\bigcup_{i=1}^n \bar{X}_i) \mid \mu[\bar{X}_i] \in r_i\}$ 

# Exercise 3.1

Prove that the Generalized Natural Join is well-defined, i.e., that the order how to join the  $r_i$  does not matter.

Proceed as follows:

- Show that the natural join is commutative,
- Show that the natural join is associative,
- ... then complete the proof.

#### 98

# **Relational Division**

Assume  $r \in \operatorname{Rel}(\bar{X})$  and  $s \in \operatorname{Rel}(\bar{Y})$  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  $\overline{Z}$ -values that occur in  $\pi[\overline{Z}](r)$ , with the additional condition that they occur in r together with each of the  $\overline{Y}$  values that occur in s".

Formally,

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

- Simple observation:  $\pi[\bar{Z}](r) \supseteq r \div s$ . This constrains the set of possible results.
- Often,  $\bar{Z}$  and  $\bar{Y}$  correspond to the keys of relations that represent the instances of entity types.
- Exercise: the explicit " $\mu \in \pi[\overline{Z}](r)$ " in the first characterization looks a bit redundant. Is it? or why not?

# Example 3.13 (Relational Division)

Compute those organizations that have at least one member on each continent:

First step: which organizations have (some) member on which continents:

*π*[organization,continent]

encompasses

SELECT DISTINCT i.organization, e.continent FROM ismember i, encompasses e WHERE i.country=e.country ORDER by 1

orgOnCont		
organization	continent	
UN	Europe	
UN	Asia	
UN	N.America	
UN	S.America	
UN	Africa	
UN	Australia	
NATO	Europe	
NATO	N.America	
NATO	Asia	
EU	Europe	
:	:	

100



• UN: occurs with each continent in orgOnCont  $\Rightarrow$  belongs to the result.

• NATO: does not occur with each continent in orgOnCont  $\Rightarrow$  does not belong to the result.

• EU: does not occur with each continent in orgOnCont  $\Rightarrow$  does not belong to the result.

# Example 3.13 (Cont'd)

Consider again the formal algebraic characterization of the division:

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

- 1. r = orgOnCont,  $s = \pi[name](continent)$ , Z = Country.
- 2.  $(\pi[\overline{Z}](r) \times s)$  contains all tuples of organizations with each of the continents, e.g., (NATO, Europe), (NATO, Asia), (NATO, N. America), (NATO, S. America), (NATO, Africa), (NATO, Australia).
- *3.*  $((\pi[\overline{Z}](r) \times s) \setminus r)$  contains all such tuples which are not "valid", e.g., (NATO,Africa).
- 4. projecting this to the organizations yields all those organizations where a non-valid tuple has been generated in (2), i.e., that have no member on some continent (e.g., NATO).
- 5.  $\pi[\bar{Z}](r)$  is the list of all organizations ...
- 6. ... subtracting those computed in (4) yields those that have a member on each continent.





103

# EXPRESSIONS IN THE RELATIONAL ALGEBRA AS QUERIES

Let  $\mathbf{R} = \{R_1, \dots, R_k\}$  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** S that maps every relation name  $R_i$  in  $\mathbf{R}$  to a relation  $S(R_i) \subseteq \text{Tup}(\bar{X}_i)$ 

Every algebra expression Q defines a **query** against the state S of the database:

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

104

# SUMMARY: INDUCTIVE DEFINITION OF EXPRESSIONS

#### Atomic Expressions

For an arbitrary attribute A and a constant a ∈ dom(A), the constant relation A : {a} is an algebra expression.

 $\Sigma_{A:\{a\}} = [A] \text{ and } S(A:\{a\}) = A:\{a\}$ 

- Every relation name  ${\it R}$  is an algebra expression.

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

# SUMMARY (CONT'D)

#### **Compound Expressions**

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

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

#### Difference

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

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

#### 106

# INDUCTIVE DEFINITION OF EXPRESSIONS (CONT'D)

#### Selection

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

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

#### Projection

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

 $\Sigma_Q = \overline{Y} \text{ and } \mathcal{S}(Q) = \pi[\overline{Y}](\mathcal{S}(Q_1)).$ 

#### Natural Join

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

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

#### Renaming

For  $\Sigma_{Q_1} = \{A_1, \dots, A_k\}$  and  $\{B_1, \dots, B_k\}$  a set of attributes,  $Q = \rho[A_1 \to B_1, \dots, A_k \to B_k](Q_1)$  is the **renaming** of  $Q_1$  $\Sigma_Q = \{B_1, \dots, B_k\}$  and  $\mathcal{S}(Q) = \rho[A_1 \to B_1, \dots, A_k \to B_k](\mathcal{S}(Q_1)).$ 

#### Example

#### Example 3.15

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

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

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

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

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

Simpler expression:

```
\pi [Name, CName] ((Professor \bowtie (teach \setminus examine)) \bowtie Course)
```

108

# **EQUIVALENCE OF EXPRESSIONS**

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

Equivalence of expressions is the basis for algebraic optimization.

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

#### Projections

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

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

# Selections

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

# 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

- $\operatorname{attr}(\alpha) \subseteq \bar{X}_1 \cap \bar{X}_2 \Rightarrow \sigma[\alpha](Q_1 \bowtie Q_2) \equiv \sigma[\alpha](Q_1) \bowtie \sigma[\alpha](Q_2).$
- $\operatorname{attr}(\alpha) \subseteq \bar{X}_1, \operatorname{attr}(\alpha) \cap \bar{X}_2 = \emptyset \Rightarrow \sigma[\alpha](Q_1 \bowtie Q_2) \equiv (\sigma[\alpha](Q_1)) \bowtie Q_2.$
- Assume  $\overline{V} \subseteq \overline{X_1 X_2}$  and let  $\overline{W} = \overline{X_1} \cap \overline{VX_2}$ ,  $\overline{U} = \overline{X_2} \cap \overline{VX_1}$ . Then,  $\pi[\overline{V}](Q_1 \bowtie Q_2) \equiv \pi[\overline{V}](\pi[\overline{W}](Q_1) \bowtie \pi[\overline{U}](Q_2))$ ; (Note: unary operations bind stronger than binary operations)
- $\bar{X}_2 = \bar{X}_3 \Rightarrow Q_1 \bowtie (Q_2 \text{ op } Q_3) \equiv (Q_1 \bowtie Q_2) \text{ op } (Q_1 \bowtie Q_3) \text{ where op } \in \{\cup, \setminus\}.$ (distributivity of  $\bowtie$  wrt.  $\cup$  and  $\setminus$ ) Note the similarity to the arithmetic term algebra:  $n \cdot (a \pm b) = (n \cdot a) \pm (n \times b)$

# **Exercise 3.2** Prove some of the equalities (use the definitions given on the "Base Operators" slide).

#### 110

# EXPRESSIVE POWER OF THE ALGEBRA

# **Transitive Closure**

The transitive closure of a binary relation R, denoted by  $R^*$  is defined as follows:

$$\begin{aligned} R^1 &= R \\ R^{n+1} &= \{(a,b) | \text{ there is an } s \text{ s.t. } (a,x) \in R^n \text{ and } (x,b) \in R \} \\ R^* &= \bigcup_{1..\infty} R^n \end{aligned}$$

Examples:

- child(x,y): child\* = descendant
- flight connections
- flows\_into of rivers in MONDIAL

# Theorem 3.2

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)

# Aspects

- join as "extending" operation (cartesian product "all pairs of X and Y such that ...")
- equijoin as "restricting" operation
- natural join/equijoin in many cases along key/foreign key relationships
- relational division (in case of queries of the style "return all X that are in a given relation with all Y such that ...")

112

# 3.2 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, INGRES@Univ. Berkeley, first product: Oracle in 1978)

# 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 (triggers)
- recursion
- object-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 → Table
   Tuple → Row

Attribute  $\rightsquigarrow$  Column

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

 comprehensive treatment in the "Practical Training SQL" (http://dbis.informatik.uni-goettingen.de/Teaching/DBP/)

#### 114

# BASIC STRUCTURE OF SQL QUERIES

SELECT  $A_1, \ldots, A_n$ 

- (... corresponds to  $\pi$  in the algebra)
- FROM  $R_1, \ldots, R_m$  (... specifies the contributing relations)
- WHERE F (... corresponds to  $\sigma$  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  $\rightarrow$  prefixing with or without aliasing (optional)

#### Example

```
SELECT code, capital, country.population, city.population
FROM country, city
WHERE code = country -- only country has a code
AND city.name = capital -- city and country have names
AND city.province = country.province;
```

- SQL is case-insensitive, i.e. CITY=city=City=cItY.; inside quotes not case-insensitive, i.e. City='Berlin' ≠ City='berlin',
- String constants: use single quotes,
- end of command: semicolon ";",
- comment syntax: inside "/\* ... \*/" or from "--" to end of line.

# PREFIXING, ALIASING AND RENAMING

- Prefixing: tablename.attr
- Aliasing of relations in the FROM clause:

SELECT  $alias_1.attr_1, alias_2.attr_2$ FROM  $table_1 \ alias_1, \ table_2 \ alias_2$ WHERE ...

Renaming of result columns of queries:
 SELECT attr<sub>1</sub> AS name<sub>1</sub>, attr<sub>2</sub> AS name<sub>2</sub>
 FROM ... WHERE ...
 (formal algebra equivalent: renaming)

116

# SUBQUERIES

Subqueries of the form (SELECT ... FROM ... WHERE ...) can be used anywhere where a relation is required:

Subqueries in the FROM clause allow for selection/projection/computation of intermediate results/subtrees before the join:

```
SELECT ...

FROM (SELECT ... FROM ... WHERE ...),

(SELECT ... FROM ... WHERE ...)

WHERE ...
```

(interestingly, although "basic relational algebra", this has been introduced e.g. in Oracle only in the early 1990s.)

Subqueries in other places allow to express other intermediate results:

```
SELECT ... (SELECT ... FROM ... WHERE ...) FROM ...
WHERE [NOT] value1 IN (SELECT ... FROM ... WHERE)
AND [NOT] value2 comparison-op [ALL|ANY] (SELECT ... FROM ... WHERE)
AND [NOT] EXISTS (SELECT ... FROM ... WHERE);
```

# SUBQUERIES IN THE FROM CLAUSE

• often in combination with aliasing and renaming of the results of the subqueries.

```
SELECT alias1.name1,alias2.name2
FROM (SELECT attr1 AS name1 FROM ... WHERE ...) alias1,
    (SELECT attr2 AS name2 FROM ... WHERE ...) alias2 WHERE ...
... all big cities that belong to large countries:
SELECT city, country
FROM (SELECT name AS city, country AS code2
    FROM city
    WHERE population > 1000000
    ),
    (SELECT name AS country, code
    FROM country
    WHERE area > 1000000
    )
WHERE code = code2;
```

118

# **SUBQUERIES**

• Subqueries of the form (SELECT ... FROM ... WHERE ...) that result in a single value can be used anywhere where a value is required

SELECT function(..., (SELECT ... FROM ... WHERE ...))
FROM ...;
SELECT ...
FROM ...
WHERE value1 = (SELECT ... FROM ... WHERE ...)
AND value2 < (SELECT ... FROM ... WHERE ...);</pre>

Subqueries in the WHERE clause		
Non-Correlated subqueries		
. the simple ones. Inner SFW independent from outer SFW		
SELECT name FROM country WHERE area > (SELECT area FROM country	SELECT name FROM country WHERE code IN (SELECT country FROM encompasses	
WHERE code='D'); Correlated subqueries	WHERE continent='Europe');	
Inner SELECT FROM WHERE referer	nces value of outer SFW in its WHERE clause:	
<pre>SELECT name FROM city WHERE population &gt; 0.25 *   (SELECT population    FROM country    WHERE country.code = city.country)</pre>	<pre>SELECT name, continent FROM country, encompasses enc WHERE country.code = enc.country AND area &gt; 0.25 * (SELECT area FROM continent ; WHERE name = enc.continent);</pre>	

120

#### Subqueries: EXISTS

• EXISTS makes only sense with a correlated subquery:

```
SELECT name
FROM country
WHERE EXISTS (SELECT *
        FROM city
        WHERE country.code = city.country
        AND population > 1000000);
```

algebra equivalent: semijoin.

• NOT EXISTS can be used to express things that otherwise cannot be expressed by SFW:

```
SELECT name

FROM country

WHERE NOT EXISTS (SELECT *

FROM city

WHERE country.code = city.country

AND population > 1000000);
```

Alternative: use (SFW) MINUS (SFW)

# SET OPERATIONS: UNION, INTERSECT, MINUS/EXCEPT

(SELECT name FROM city) INTERSECT (SELECT name FROM country);

Often applied with renaming:

```
SELECT *
FROM ((SELECT river AS name, country, province FROM geo_river)
        UNION
        (SELECT lake AS name, country, province FROM geo_lake)
        UNION
        (SELECT sea AS name, country, province FROM geo_sea))
WHERE country = 'D';
```

```
122
```

#### Set Operations and Attribute Names

The relational algebra requires  $\bar{X} = \bar{Y}$  for  $R(\bar{X}) \cup S(\bar{X})$ ,  $R(\bar{X}) \cap S(\bar{X})$ , and  $R(\bar{X}) \setminus S(\bar{X})$ :

• attributes are unordered, the tuple model is a "slotted" model.

In SQL,

(SELECT river, country, province FROM geo\_river)
UNION
(SELECT lake, country, province FROM geo\_lake)

is allowed and the resulting table has the format (river, country, province) (note that the name of the first column may be indeterministic due to internal optimization).

- the SQL model is a "positional" model, where the name of the *i*-th column is just inferred "somehow",
- cf. usage of column number in ... ORDER BY 1,
- note that column numbers can only be used if there is no ambiguity with numeric values, e.g., SELECT name, 3 FROM country yields a table whose second column has always the value 3.

# SYNTACTICAL SUGAR: JOIN

- basic SQL syntax: list of relations in the FROM clause, cartesian product, conditions in the WHERE clause.
- explicit JOIN syntax in the FROM clause:

SELECT ...

FROM  $R_1$  NATURAL JOIN  $R_2$  ON  $join-cond_{1,2}$  [NATURAL JOIN  $R_3$  ON  $join-cond_{1,2,3}$  ...] WHERE ...

- usage of parentheses is optional,
- same translation to internal algebra.

# **OUTER JOIN**

- Syntax as above, as LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN (and FULL JOIN, which is equivalent to FULL OUTER JOIN).
- usage of parentheses is optional, otherwise left-first application (!).
- can be translated to internal outer joins, much more efficient than handwritten outer join using UNION and NOT EXISTS.

124

# HANDLING OF DUPLICATES

In contrast to algebra relations, SQL tables may contain duplicates (cf. Slide 114):

- some applications require them
- duplicate elimination is relatively expensive ( $O(n \log n)$ )
- $\Rightarrow$  do not do it automatically
- $\Rightarrow$  SQL allows for *explicit* removal of duplicates:

Keyword: SELECT DISTINCT  $A_1, \ldots, A_n$  FROM ...

The internal optimization can sometimes put it at a position where it does not incur additional costs.

# GENERAL STRUCTURE OF SQL QUERIES:

SELECT [DISTINCT]  $A_1, \ldots, A_n$ FROM  $R_1, \ldots, R_m$ WHERE FGROUP BY  $B_1, \ldots, B_k$ HAVING GORDER BY H list of expressions list of relations condition(s) list of grouping attributes condition on groups, same syntax as WHERE clause sort order – only relevant for output

• ORDER BY: specifies output order of tuples

SELECT name, population FROM city;

full syntax: ORDER BY *attribute-list* [ASC|DESC] [NULLS FIRST|LAST] (ascending/descending) Multiple attributes allowed:

SELECT \* FROM city ORDER BY country, province;

Next: How many people live in the cities in each country?

- GROUP BY: form groups of "related" tuples and generate one output tuple for each group
- HAVING: conditions evaluated on the groups

OTIM AND MEN MAY COTINT

126

#### Grouping and Aggregation • First Normal Form: all values in a tuple are atomic (string, number, date, ...) • GROUP BY attr-list: forms groups of tuples that have the same values for attr-list SELECT country, SUM(population), MAX(population), COUNT(\*) FROM City : **GROUP** BY country Innsbruck Tirol 118000 Α HAVING SUM(population) > 10000000; 1761738 Vienna Α Vienna • each group yields one tuple which may contain: : : : : - the group-by attributes Α Graz Steiermark 238000 - aggregations of all values in a column:

SUM, AVG, MIN, MAX, COUNT				
	:	:	:	:
	country: A	SUM(population): 2862618	MAX(population):1761738	COUNT(*):13
	country: D	SUM(population):25333235	MAX(population):3292365	COUNT(*):85
	:	:	:	:

2

:

1

• HAVING (condition on groups) AND SELECT use these values.

#### Aggregation

Aggregation can be applied to a whole relation:

SELECT COUNT(\*), SUM(population), MAX(population)
FROM country;

• Aggregation with DISTINCT:

SELECT COUNT (DISTINCT country) FROM CITY WHERE population > 1000000;

128

# **ALTOGETHER: EVALUATION STRATEGY**

list of expressions
list of relations
condition(s)
list of grouping attributes
condition on groups, same syntax as WHERE clause
sort order – only relevant for output

- 1. evaluate FROM and WHERE,
- 2. evaluate GROUP BY  $\rightarrow$  yields groups,
- 3. generate a tuple for each group containing all expressions in HAVING and SELECT,
- 4. evaluate HAVING on groups,
- 5. evaluate SELECT (projection, removes things only needed in HAVING),
- 6. output result according to ORDER BY.

# **CONSTRUCTING QUERIES**

For each problem there are multiple possible equivalent queries in SQL (cf. Example 3.14). The choice is mainly 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

130

# COMPARISON

SQL:

SELECT  $A_1, \ldots, A_n$  from  $R_1, \ldots, R_m$  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 tuples  $t_1, \ldots, t_n$  satisfy the WHERE-clause THEN

evaluate the SELECT clause and generate the result tuple (projection).

Note: the tuple variables can also be introduced in SQL explicitly as alias variables:

SELECT  $A_1, \ldots, A_n$  FROM  $R_1$   $t_1, \ldots, R_m$   $t_m$  WHERE F (then optionally using  $t_i.attr$  in SELECT and WHERE)

#### Comparison: Subqueries

• Subqueries in the FROM-clause (cf. Slide 118): joined subtrees in the algebra



• the relation from evaluating the from clause has columns city, code2, country, code that can be used in the where clause and in the select clause.





Note that the natural semijoin serves as an equi-selection where all tuples from the outer expression qualify that match an element of the result of the inner expression.

### Comparison: Subqueries





#### Comparison: Correlated Subqueries

- ... comment to previous slide:
  - although the tree expression looks less target-oriented than the SQL correlated subquery, it does the same:
  - instead of iterating over the tuples of the outer SQL expression and evaluating the inner one for each of the tuples,
  - the results of the inner expression are "precomputed" and iteration over the outer result just fetches the corresponding one.
  - · effectiveness depends on the situation:
    - how many of the results of the subquery are actually needed (worst case: no tuple survives the outer local WHERE clause).
    - are there results of the subquery that are needed several times.

database systems are often able to internally choose the most effective solution (schema-based and statistics-based)

... see next section.

136

# Comparison: EXISTS-Subqueries

- WHERE EXISTS: similar to above: correlated subquery, no additional condition after natural semijoin
- SELECT ... FROM X,Y,Z WHERE NOT EXISTS (SFW):

```
SELECT ...
FROM ((SELECT * FROM X,Y,Z) MINUS
(SELECT X,Y,Z WHERE EXISTS (SFW)))
```

#### Results

- all queries (without NOT-operator) including subqueries without grouping/aggregation can be translated into SPJR-trees (selection, projection, join, renaming),
- they can even be flattened into a single broad cartesian product, followed by a selection and a projection,
- so-called "SPJR-algebra" or "conjunctive queries", whose optimization plays an important role.

Comparison: the differences between Algebra and SQL

- The relational algebra has no notion of grouping and aggregate functions. Such operators can be defined as additional base operators (see Exercises)
- SQL has no clause that corresponds to relational division.
   Such queries must be constructed by the users, using the existing SQL constructs.

# Example 3.16 (Relational Division in SQL)

Consider again Example 3.13 (Slide 100):

"Compute those organizations that have at least one member on each continent":

 $orgOnCont \div \pi[name](continent).$ 

# Exercise (DIV-1):

Use your commonsense logical reasoning to express this query in SQL.

**Exercise (DIV-2):** Use the algebraic expression for  $r \div s$  from Slide 99 for stating the query in SQL (use the SQL statement for orgOnCont from Slide 100):

 $r \div s = \pi[\bar{Z}](r) \setminus \pi[\bar{Z}]((\pi[\bar{Z}](r) \times s) \setminus r).$ 

(try both now before continuing with the video)

138

# Example 3.16 (Cont'd: Commonsense logical reasoning – Brain 1.0)

The relational division corresponds to the universal quantifier – "such that all ..." or "such that each of ...":

- "... those organizations o such that that for each continent c, there is some country x such that x is a member of o and x is located on c. (here, x is a "witness")
- "... those organizations such that there is <u>no</u> continent c such that there is <u>no</u> country x such that x is a member of o and x is located on c.

(... no continent where no such witness can be found)

can be expressed by not exists - not exists:

```
select abbreviation
from organization o
where not exists
(select *
  from continent c
  where not exists
  (select *
    from country x
    where (x.code, o.abbreviation) in (select country, organization from ismember)
        and (x.code, c.name) in (select country, continent from encompasses)
    ))
```

```
Example 3.16 ((DIV-2) – inserting into r \div s = \pi[\bar{Z}](r) \setminus \pi[\bar{Z}]((\pi[\bar{Z}](r) \times s) \setminus r))
 (select org
  from (select distinct i.organization as org, e.continent as cont
        from ismember i, encompasses e
        where i.country = e.country ))
                                              Nobody would do this:
minus

    learn this formula,

 ( select o1
   from ((select o1,n1

    copy&paste and fight with parentheses!

           from (select org as o1
                 from (select distinct i.organization as org, e.continent as cont
                        from ismember i, encompasses e
                        where i.country = e.country )
                ),
                (select name as n1 from continent)
         )
         minus
         (select distinct i.organization as org, e.continent as cont
          from ismember i, encompasses e
          where i.country = e.country )
        )
 )
```

```
140
```

# Example 3.16 (Cont'd)

 Instead of π[Z](r), a simpler query yielding the Z values can be used. These often correspond to the keys of some relation that represents the instances of some entity type (here: the organizations):

 $orgOnCont \div \pi[name](continent) =$ 

 $\pi[abbreviation](organization) \setminus$ 

 $\pi[\overline{Z}]((\pi[abbreviation](organization) \times \pi[name](continent)) \setminus orgOnCont)$ 

 $orgs \times conts$ 

the "missing" pairs

organizations that have a missing pair

• the corresponding SQL query is much smaller, and can be constructed intuitively:

```
(select abbreviation from organization)
minus ... the structure is the same as the previous one!
( select abbreviation
  from ((select o.abbreviation, c.name
     from organization o, continent c)
     minus
     (select distinct i.organization as org, e.continent as cont
     from ismember i, encompasses e
     where i.country = e.country ) ) )
```

# Example 3.16 (Cont'd)

The corresponding SQL formulation that implements division corresponds to the textual

"all organizations that occur in orgOnCont together with each of the continent names",

or equivalently

"all organizations org such that there is no value cont in  $\pi[name](continent)$  such that org does not occur together with cont in orgOnCont".

```
select abbreviation
from organization o
where not exists
((select name from continent)
  minus
  (select cont
    from (select distinct i.organization as org, e.continent as cont
        from ismember i, encompasses e
        where i.country = e.country )
    where org = o.abbreviation))
```

• the query is still algebra-style set-theory-based.

Oracle Query Plan Estimate: not-exists-not-exists: 339; copy-and-paste-solution: 707; minus-minus: 20; not-exists-minus: 341.

142

# Example 3.16 (Cont'd)

Aside: logic-based querying with Datalog (see Lecture on "Deductive Databases") corresponding to the minus-minus solution:

 $\{o \mid \textit{organization}(o, ...) \land \neg \exists \textit{cont} : (\textit{continent}(\textit{cont}, ...) \land \neg \textit{orgOnCont}(o, \textit{cont}))\}$ 

```
 \begin{array}{l} & \mbox{[mondial].} \\ & \mbox{orgOnCont(0,C,Cont) :- isMember(C,0,_), encompasses(C, Cont,_).} \\ & \mbox{notResult(0) :- organization(0,_,,_,_), continent(Cont,_), not orgOnCont(0,_,Cont).} \\ & \mbox{result(0) :- organization(0,_,,_,_), not notResult(0).} \\ & \mbox{''- result(0).} \\ & \mbox{''- result(0).} \\ & \mbox{''- findall(0, result(0), L).} & [Filename: Datalog/orgOnContsDiv.P] \\ & \mbox{... much shorter.} \\ & \mbox{Algebra expression for it:} \\ & \mbox{$\pi$[abbrev](org) $\times$ $\pi$[name](cont) $ $\rho$[org$$-$abbrev]($\pi$[org$,cont](isMember $\bowtie$ encompasses)) $ \end{tabular} } \end{array}
```

corresponds to the most efficient minus-minus solution.
#### 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
- subqueries in the SELECT clause (returning a single value)
- · 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.

144

# 3.3 Efficient Algebraic Query Evaluation

Semantical/logical optimization: Consider integrity constraints in the database.

•	• constraint on table city: $population \ge 0$ .									
	Query plan for select $*$ from city where population $< 0$ :									
	Operation	object	pred	icate	cost					
	SELECT STATEMENT				0					
	_FILTER		NUL	L IS NOT NULL						
	TABLE ACCESS (FULL)	CITY	POP	ULATION < 0	7					
(foreign key references activated)										
	select * from ismember wh	ere cou	ntry 1	not in (select	code	from	country):			
	Operation	object		predicate		cost				
	SELECT STATEMENT					0	-			
	_FILTER			NULL IS NOT N	IULL					
	TABLE ACCESS (FULL)	ISMEM	BER			9				
						-				

Semantical/logical optimization (Cont'd): Consider integrity constraints in the database.

(foreign key references activated)

select country from ismember where country in (select code from country):

OperationobjectpredicatecostSELECT STATEMENT9\_TABLE ACCESS (FULL)ISMEMBER9

No lookup of country.code at all (because guaranteed by foreign key)

- not always obvious
- general case: first-order theorem proving.

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.

146

## **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 → 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





## **OPTIMIZATION BY TREE RESTRUCTURING**

- Equivalent transformation of the operator tree that represents an expression
- · Based on the equivalences shown on Slide 109.
- 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
- · semijoins instead of joins (in combination with implementation issues; see next section)

Push Selections DownAssume  $r, s \in \operatorname{Rel}(\bar{X}), \bar{Y} \subseteq \bar{X}$ . $\sigma[cond](\pi[\bar{Y}](r)) \equiv \pi[\bar{Y}](\sigma[cond](r))$ <br/>(condition: cond does not use attributes from  $\bar{X} - \bar{Y}$ ,<br/>otherwise left term is undefined) $\sigma_{pop>1E6}(\pi[name, pop](country)) \equiv \pi[name, pop](\sigma_{pop>1E6}(country))$ <br/> $\sigma[cond](r \cup s) \equiv \sigma[cond](r) \cup \sigma[cond](s)$ <br/> $\sigma_{pop>1E6}(\pi[name, pop](country) \cup \pi[name, pop](city))$ <br/> $\equiv \sigma_{pop>1E6}(\pi[name, pop](country)) \cup \sigma_{pop>1E6}(\pi[name, pop](city))$ <br/> $\sigma[cond](\rho[N](r)) \equiv \rho[N](\sigma[cond'](r))$ <br/>(where cond' is obtained from cond by renaming according to N)<br/> $\sigma[cond](r - s) \equiv \sigma[cond](r) - \sigma[cond](s)$ <br/> $\sigma[cond](r - s) \equiv \sigma[cond](r) - \sigma[cond](s)$ <br/> $\pi$  : see comment above. Optimization uses only left-to-right.

#### Push Selections Down (Cont'd)

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

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

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

Then,

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

#### Example 3.17

Names of all countries that have an area of more than 1,000,000 km<sup>2</sup>, their capital has more than 1,000,000 inhabitants, and more than half of the inhabitants live in the capital.  $\Box$ 

152



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

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  $\bar{X} \setminus Y$ .

 $\pi[\bar{Y}](\sigma[cond](r)) \equiv \sigma[cond_{\bar{Y}}](\pi[\bar{Y}](\sigma[cond_{\bar{X}}](r)))$  $\pi[\bar{Y}](\rho[N](r)) \equiv \rho[N](\pi[\bar{Y}'](r))$ 

(where  $\bar{Y}'$  is obtained from  $\bar{Y}$  by renaming according to N)

 $\pi[\bar{Y}](r \cup s) \qquad \equiv \quad \pi[\bar{Y}](r) \cup \pi[\bar{Y}](s)$ 

- Note that this does *not* hold for "∩" and "-"!
- advantages of pushing " $\sigma$ " vs. " $\pi$ " are data-dependent Default: push  $\sigma$  lower.

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

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

· complex interactions between reusing subexpressions and pushing selection/projection

154



• evaluation: semijoin uses an index (on the key of ismember) or nested-loop.

Application Order of Joins Minimize intermediate results (and number of comparisons): ... consider the equivalent query: SELECT organization.name as org, country.name as cname FROM organization, isMember, country WHERE organization.abbreviation = isMember.organization AND isMember.country = country.code If primary key and foreign key indexes on country.code and organization.abbreviation are available:  $\pi$ [oname,cname] (10000) loop over isMember | (10000) extend each tuple with matching (10000) 🖂  $\rho$ [name $\rightarrow$ cname, organization and country by using code→country] the indexes.

 Oracle query plan shows an extremely efficient evaluation of both of the above queries using indexes and ad-hoc views.



156

Aside: the real query p	lan				
(see Slide 160 ff. for deta	ails)				
Operation	Óbiect	Pred(Index)	Pred(Filter)	COST	Rows
				1000	110W3
SELECT STATEMENT				13	9968
_HASH JOIN		C.CODE=ISM.COUNTRY		13	9968
VIEW	v2			2	241
HASH JOIN		ROWID=ROWID			
INDEX (FULL SCAN)	COUNTRYKEY			1	241
INDEX (FULL SCAN)	SYS_C0030486			1	241
HASH JOIN		ORG.ABBREV=ISM.ORG		11	9968
VIEW	v1			2	152
HASH JOIN		ROWID=ROWID			
INDEX (FULL SCAN)	ORGKEY			1	152
INDEX (FULL SCAN)	ORGNAMEUNIQ			1	152
SORT (UNIQUE)				9	9968
INDEX (FULL SCAN)	MEMBERKEY			9	9968

No access to actual tables, ism(org,country) from key index, org(abbrev,name) from indexes via rowid-join, country(code,name) from indexes via rowid-join; both materialized as ad-hoc-views, combined by two hash-joins.

# **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 3.18 • $\sigma$ [country = "D" $\land$ population > 200000](City): Assume an index that supports the condition country = "D". - without pipelining: compute $\sigma$ [country = "D"](City) using the index, obtain City'. Then, *compute* $\sigma$ [population > 200000](City'). - with pipelining: compute $\sigma$ [country = "D"](City) using the index, and check **on-the fly** each qualifying tuple against $\sigma$ [population > 200000]. - extreme case: when there is also an index on population (tree index, allows for range scan): obtain set $S_1$ of all tuple-ids for german cities from index on code, obtain set $S_2$ of all tuple-ids of cities with more than 2 million inhabitants from population index, intersect $S_1$ and $S_2$ and access only the remaining cities.

#### Pipelining

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

158

- $\sigma[cond](R \bowtie S)$ :
  - 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 combine materialization of intermediate results, iterator-based implementation of algebra operators, indexes, and pipelining.

# Chapter 4 Internal Organization and Implementation

This section heavily relies on other subdisciplines of Practical Computer Science:

- System Structures, down to the physical level
- Operating Systems Aspects: Caching
- Algorithms (mainly: for joins) and Data Structures (tree indexes, hashing)

160

## **PHYSICAL DATA ORGANIZATION**

- the **conceptual schema** defines which data is described and its semantics.
- the **logical schema** defines the actual relation names with their attributes (and datatypes), keys, and integrity constraints.
- the physical schema defines the physical database where the data is actually stored.
   ⇒ 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]
- for now, it is sufficient to note that pagewise access has to be dealt with.

#### 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 a **directory** that assigns a physical address to each n.

- memory management for deleted records
- · different strategies for fixed-length and variable-length records

Simplified storage of a page of the Country table:

L								/													
1	Η	5	Ι	A	a	F	X	/ir	nt: 90	2123	80		b	pigint	: 57	460	274			4	R
0	m	е	5	L	а	z	i	0	2	С	Æ	Ţ	S	w	i	t	z	е	r	Ι	а
n	d	/il	nt: 4	129	0		bigint: 7207060 4 B e r n 2 B							Е							
1	B	7	В	е	Ι	g	i	u	u m int: 30510 bigint: 10170241												
8	В	r	u	s	s	е	Ι	s	7	В	r	а	b	а	n	t	•	•			
so far to the physical facts																					

164

# 4.1 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
- in general, joins by key/foreign-key references are supported by indexes.

166

## **TREE INDEXES**

This topic brings data structures and databases (= applications of data structures) together.

- introductory lecture "Computer Science I": store numbers in trees.
- databases: tree *index* over the values of a column of a relation
  - search tree based on the values (numbers, strings)
  - the tuples themselves are not stored in the tree
  - entries (or leaf entries only) hold the values and point to the respective tuples in the database
- special trees with higher degrees:
  - each node (of the size of a storage page) has multiple entries and multiple children.



## Example: BSB-based TreeSet in SQL Query Answering

[Exercise/Demonstrate on whiteboard]

- (1) SELECT country FROM City WHERE population > 1000000; contains duplicates.
- (2) SELECT distinct country FROM City WHERE population > 1000000;
  - initialize an empty TreeSet rs,
  - evaluate (1),
  - during computation, for each result r:
  - if  $r \notin rs$ , then add it, and output r;
  - requires  $n \cdot \log n$  steps.
  - sketch Java class: treenode(left: node, right: node, value: String)
  - sketch representation of BSB in storage page/array (cf. CS I) [node at position  $n \rightarrow$  left child at position 2n, right child at position 2n + 1]
- (3) SELECT country, count(\*) FROM City WHERE population > 1000000 GROUP BY country;
  - same strategy as above, additional data (count) stored in the tree,
  - if  $r \notin rs$ , then add (r, count : 1) it, and output r,
  - if  $r \in rs$ , then increment count of r.

## **B- AND B\*-TREES**

A **B-tree** (R. Bayer & E.McCreight, 1970) of order  $(m, \ell)$ ,  $m \ge 3$ ,  $\ell \ge 1$ , 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), and hold at most  $\ell$  entries
- inner nodes have the form  $(p_0, k_1, p_1, k_2, p_2, \dots, k_n, p_n)$  where  $\lceil m/2 \rceil 1 \le n \le m 1$  and
  - $k_i$  are search key values, ordered by  $k_i < k_j$  for i < j
  - $p_i$  points to the i + 1th 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)

B-trees are used for "simply" organizing items of an ordered set (e.g. for sorting) as an extension of binary search trees.

170

## **B\*-TREES**

(sometimes also called B<sup>+</sup>-Trees)

A **B**\*-tree of order  $(m, \ell)$ ,  $m \ge 3$ ,  $\ell \ge 1$ , is closely related, except:

- they are intended to associate *data* with search key values:
- the inner nodes do not hold additional data, but are still intended to guide the search. (organized internally e.g. as binary search trees)
- The leaves are of the form  $([k_1, s_1], [k_2, s_2], \ldots, [k_g, s_g])$  where  $g \leq \ell$ ,  $k_i$  is a search key value, and  $s_i$  is a data record or (in databases) a pointer to the corresponding record.



#### Properties

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

#### Use of B\*-Trees as Access Paths in Databases

- databases: cities stored in data files, index trees hold *pointers* to city records in their tree entries.
- separation between index files/pages and data files/pages.
- multiple search trees for each relation possible.

#### Example

#### Example 4.1

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

- every leaf (tuple) page contains 10 cities,
- every inner node contains 20 pointers

#### Then

- every inner node on the lowest level covers 200 cities
- every inner node on the second lowest level covers 4000 cities
- minimal: only one level of inner nodes
- maximal: two levels of inner nodes (nodes about only 2/3 filled)
- access every city with WHERE Name =" ... " in 3 or 4 steps
- index on population, e.g., for WHERE Population > 1,000,000 ORDER BY Population
- realistic numbers: block size 4K: lowest level (keys+pointers to DB): 100 cities; inner nodes: 100 references.

174

## HASH-INDEX (DICTIONARY)

Hash index over the value of one or more columns ("hask key" – is not necessarily a key of the relation):

The values are distributed over k tiles.

- A hash function h is a function that maps each value to a tile number. operation: lookup(value)
- each tile holds pairs (key,pointer) to all tuples whose hash-key value is mapped to this tile;
- · each tile consists of one or more pages.
- common technique: convert value to an integer i.  $i \mod k$  gives the tile number.



#### Hash-Index

#### Example 4.2

Multi-attribute hash keys:

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

#### Comments:

- · maintenance of overflow pages: see Info III
- lookup inside each tile can be organized by a B-tree (kept on a single page)

## TREE AND HASH INDEXES: OBSERVATIONS

- structure of tree leaf nodes and hash tiles is the same:
  - pairs: (search key value, reference to a tuple in storage page)
- tree leaf nodes and hash tiles are actually *projections* of a table R on the search key attribute/attributes

-  $\pi$ [name](City) can be answered by just using the above tree/hash indexes.

 $\Rightarrow$  indexes not only be as access paths, but also to support frequently used projections.

### **DUPLICATES IN TREE OR HASH INDEXES**

What if  $\pi$ [search-key-attrs](relation) contains duplicates,

e.g., index on City.country?

- straightforward: multiple subsequent (value, ref) entries for the same value in the leaf/tile pages,
- saves some space: pairs (value, set-of-references).
- any index that is not over a superset of a candidate key potentially contains duplicates.

178

## **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 *a* exist. Then, a bit list index to *f* consists of *k* bit vectors  $B(v_i)$ ,  $i \le 1 \le k$ .

•  $B(v_i)(j) = 1$  if the *j*th 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 4.3

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

isMember								
Org.	Country	Туре						
EU	D	member						
UN	D	member						
UN	СН	observer						
UN	R	member						
EU	PL	applicant						
EU	CZ	applicant						
:	:	:						

Bit list for type=member:	110100
Bit list for type=observer:	001000
Bit list for type=applicant:	000011
Ditlict for over 511	100011
Bit list for $org=EO$ :	100011
Bit list for org=UN:	011100

- 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.
- 2nd bit list on organization column: "members of UN" by logical "and" of bit lists.

180

# ORDERED STORAGE

Tuples of a relation can be stored grouped/ordered wrt. one search key

• Example: table City grouped by country (or even by (country, province))

Relation "City"

Vienna	A	1583000
Innsbruck	A	118000
Bern	СН	134393
Berlin	D	3472009
Hamburg	D	1705872
Munich	D	1244676
Hannover	D	525763
Göttingen	D	127519
Aachen	D	247113
Paris	Ц	2152423
Grenoble	Ц	150758
Lisbon	Р	2063800

- things that are frequently needed together can be fetched within the same page,
- Index on City.country needs only a reference to the first tuple in each country (note that the index is still useful to have access in  $O(\log n)$  or O(1)).
- insertions and modifications more costly (strategies to allow and keep free space between blocks)

### CLUSTERING

- keep data that semantically belongs together on the same pages
- obviously done for relations, by ordered storage even inside a relation
- 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  $\bar{Y}\mbox{-values}.$
  - provides obvious advantages when evaluating joins over the cluster key.

#### 182

#### Example 4.4

Consider the relation schemata	Organiza	ation	
and	Abbrev		
isMember(organization, country, type).	EU	Name	established
The foreign key		European Union	07.02.1992
isMember.organization $\rightarrow$ organiza-		Country	Туре
tion.abbreviation		D	member
is used as cluster key.		A	member
		:	:
	UN	Name	established
		United Nations	26.06.1945
		Country	Туре
		D	member
		СН	observer
		:	:

## 4.1.1 Algorithms and Data Structures: 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).

Indexes on single attributes: obvious.

#### Indexes on multiple attributes:

- 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**  $\theta$  **value** can be used 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 (Country,Province,CityName) is the search key of an index, it can also be used as an index for the key's prefix (Country,Province).

184

# 4.1.2 Implementation of 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*  $\theta$  value" (binary search) and process these tuples.
- tree index: find the tuple(s) that satisfy "*field*  $\theta$  value" using the tree index and process these tuples.
- hash: suitable only if  $\theta$  is equality.

#### Selection: boolean combination of predicates

Boolean combinations of predicates can be evaluated by set operations on references (from the tree leaves and hash tiles):

Consider  $\pi$ [name]( $\sigma$ [country="D"  $\wedge$  population > 500.000](City))

- hash index on City.country,
- tree index on City.population (supports range search for > 500.000)

Leaf entries and entries on hash tiles are of the form

(search-key-value, reference-to-tuple)

- (i) hash lookup on City.country="D" results in a set of references,
- (ii) index search on City.population>500.000 results in a set of references.
  - compute intersection:
  - put (i) into a TreeSet or HashSet s (fits in main memory)
  - for each reference in (ii): if it is contained in s, access actual tuple and output its name.

186

# PROJECTION

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

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

- if an index over *field*<sub>1</sub>, *field*<sub>2</sub>, ..., *field*<sub>m</sub> or a superset of it exists: scan only the index leaf nodes and apply a projection to them.
- 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. 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.

 $\Rightarrow$  hashing not only as an index for search *structures*, but also as an *algorithm* for partitioning.

## JOIN

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

Nested-loop-join

for each tuple  $r \in R$  do

foreach tuple  $s\in S$  do

if  $r_A = s_B$  then add [r, s] to result

Very inefficient:

- assumption: only one page of each relation fits into main memory
- *m* tuples of *R* per page, *n* tuples of *S* per page:  $|R|/m + |R| \cdot |S|/n$  filesystem accesses (for each tuple of *R*, loop over all pages of *S*).
- $|R| \cdot |S|$  comparisons.

obvious: if possible, process the smaller relation in inner loop to keep it in main memory (|R|/m + |S|/n filesystem accesses).

188

Block-nested-loop-Join

Optimization by page-oriented strategy:

Divide S into blocks that fit in main memory and process each of the blocks separately.

- assumption: only one page of each relation fits into main memory
- m tuples of R per page, n tuples of S per page:
  - take first page  $(R_1)$  of R and first page  $(S_1)$  of S;
  - combine *each* tuple of  $R_1$  with each tuple with  $S_1$ .
  - continue with  $R_1$  and second page ( $S_2$ ) of S ... up to  $R_1$  with  $S_{last}$
  - continue with  $R_2$  and  $S_1$  etc.
- $|R|/m + |R|/m \cdot |S|/n$  filesystem accesses (for each page of *R*, loop over all *pages* of *S*).
- still  $|R| \cdot |S|$  comparisons.
- same for k > 1 available cache pages: read k 1 pages from R for each block.

#### Straightforward optimizations for Block-Nested-Loop-Join

For each block, some optimizations can be applied: (that motivate also the subsequent algorithms)

• idea: do not consider cartesian product + selection as the base for join, but look up matching tuples in *S*.

Applied to the Block-Nested-Loop-Join base algorithm, these do not reduce the number of page accesses, but the number of comparisons:

- generate a temporary index on S.B over the loaded fraction of S;
- in case of duplicates of *R*.*A*: generate a temporary index also over *R*.*A*. on the single loaded page to process tuples groupwise;
- [do not discuss yet: generate temporary ordered indexes both on *R*.*A* and *S*.*B*, process them merge-style]

next step: consider these optimizations on the global level of the algorithm

190

#### 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.

- given index on S's join attribute
- loop over R, for each value access matching tuple(s) in S.
- |R|/m + |Result| filesystem accesses,
- up to  $|R|/m + |R| \cdot |S|$  filesystem accesses,
- usually less efficient than block-nested-loop!

#### Parallelization

• Divide *R* into partitions and process each of them on a separate processor (share the index on *S*).

#### Sort-merge-join

Algorithm type: "Scan line"

First: simple case that illustrates the principle:

Assumption: relations are sorted wrt. R.A and S.B

- · search for matches as follows:
  - proceed through the ordered R and S stepwise, always doing a step in the index 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).
- |R|/m + |S|/n filesystem accesses.

#### General Case: relations are not sorted

... next slide

192

#### Sort-merge-join (cont'd)

- · Sort relations first:
  - sort *R* according to *R*.*A*:
  - copy first k pages in cache, sort them in place (Quicksort), store them,
  - do this for pages 2k...3k 1 etc.
  - requires  $2 \cdot |R|/m$  page accesses (read and write)
  - mergesort the sorted packets (linear; traverse all packets in parallel and write out in different place)
  - requires again  $2 \cdot |R|/m$  page accesses (read and write)
  - do the same for  ${\cal S}$

Sort+merge:  $4 \cdot |R|/m + 4 \cdot |S|/n + |R|/m + |S|/n$  page accesses

• [note: the step "mergesort the sorted packets" can be omitted by running merge join directly on all of them; then  $2 \cdot |R|/m + 2 \cdot |S|/n + |R|/m + |S|/n$ .]

#### Sort-merge-join (cont'd)

- Use only sorted indexes:
  - compute sorted indexes of both relations on the join attributes (fit into main memory, requires |R|/m + |S|/n) (if tree indexes on join attributes are available, simply use them)
  - search for matches as above:
    - \* proceed through the ordered indexes R and S stepwise, always doing a step in the index where the "smaller" value is.
  - if a match is found:
    - \* access actual tuples and generate a result tuple.
    - check for tuples which have the same values of the join attributes (must immediately follow this match in both relations).
  - at most |R|/m + |S|/n + |Result| filesystem accesses (when processing duplicates with same R.A = S.B value, less than + |Result|)
- with sorted indexes: costs depend on number of results ("selectivity of the join").
- $\Rightarrow$  decide to sort relations or to use indexes based on selectivity/heuristics.

194

#### Sort-merge-join: basic 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 \neq eof and Ts \neq eof do {
       while Tr \neq eof and Tr.A < Ts.B do Tr = next tuple in R after Tr;
       while Ts \neq eof and Tr.A > Ts.B do Ts = next tuple in S after Ts;
       // now, Tr.A = Ts.B: match found
       while Tr \neq eof and Tr.A = Ts.B do {
              Gs := Ts;
               while Gs \neq eof and Gs.B = Tr.A do {
                     add [Tr, Gs] to result;
                     Gs = \text{next tuple in } S \text{ after } Gs;
               Tr = \text{next tuple in } R \text{ 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.

196

## EXERCISE

#### **Exercise 4.1**

Consider the join of two relations R and S wrt. the join condition R.A = S.B. 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  $S_B$ ),

(d) sort-merge-join (sorted relation/sorted index),

(e) hash-join.

#### Another Example

In the previous comparison, the index-nested-loop-join did not perform well:

- access to each individual matching tuple is fast, but every page of  ${\cal S}$  has to be accessed n times.

#### Example 4.5

Consider  $(\sigma[cond](R)) \bowtie_{R.A=S.B} S$  where cond has a selectivity q, (e.g. q = 1/1000):

- loop over *R*, evaluate *cond*, for each tuple that satisfies the condition, access matching tuple(s) in *S* using the index.
- $|R|/m + |Result| = |R|/m + q \cdot |R| \cdot q'$  filesystem accesses,
- where q' is the selectivity of the join.
- $\Rightarrow$  usage of index is efficient when only a small number of tuples of *S* has to be accessed (the other algorithms had to process *S* completely).

П

198

## SEMIJOINS

• Natural Semijoins or Condition Semijoins as subqueries:  $R \Join [condition(R_{i_1}, ..., R_{i_1}, S_{i_1}..., S_{i_\ell})]S$ 

It is often sufficient to use an index over  $S_{i_1}...S_{i_\ell}$  instead of the whole relation.

• Semjoins as preparation for joins:  $R \bowtie S = (R \bowtie S) \bowtie S$  (and symmetrically  $= R \bowtie (R \rightarrowtail S)$ )

Is that simpler? - Sometimes Yes!

- cf. Join  $R \bowtie S$ , with an existing index over the join attribute(s) in S: iterate over R and access S via index. No access, if not found. If found, access tuple in SThis actually *is*  $(R \bowtie S) \bowtie S$ .

Consider  $R \bowtie S$  to be a hash join:

- First hash S, build an index over the join attribute(s), keep the index in main memory.
- iterate over R: for every tuple  $\mu$ , if  $\mu \in (R \bowtie S)$  (can be checked against the main memory index), then hash it.

## "PURE" INDEX-BASED SELECTIONS, PROJECTIONS, AND JOINS

Consider that leaves of tree- and hash-indexes are of the form (value, pointer-to-tuple)

- SELECT code FROM country WHERE population > 10000000 AND area > 500000
- indexes on country.population, country.area (tree) and country.code (tree or hash)
- 1. search 10000000 in the population index (yields first leaf with > 10000000),
- 2. from there, iterate over all higher entries  $(pop, ptr_1)$  in the leaves. ptr points to a tuple representing a result. Collect  $ptr_1$ s in a (Tree)Set  $S_1$ .
- 3. search 500000 in area population index (yields first leaf with > 500000),
- 4. from there, iterate over all higher entries  $(pop,ptr_2)$  in the leaves.  $ptr_2$  points to a tuple representing a result. Collect  $ptr_2$ s in a (Tree)Set  $S_2$ .
- 5. intersect  $S_1 \cup S_2$  (intersection of index *pointers*)
- 6. For each result ptr, look in the index entries for code (code,  $ptr_3$ ) for the entry with  $ptr_3 = ptr$ . Output code.
- problem: the index on *code* is looked up "backwards".

200

#### Inverted Indexes

Sometimes (as in the previous example) is would be useful to use an index backwards:

- Recall: leaves of tree- and hash-indexes are of the form (value, pointer-to-tuple)
- if the *pointer-to-tuple* is known from an index lookup, the *value* of an indexed attribute of this tuple is stored in the index leaf node.
- instead of accessing the (large) tuple(s) from its storage, it might be cheaper to lookup the values in the index.
- $\Rightarrow$  an **Inverted index** on an attribute *attr* is a (tree or hash) index for search by the tuple-pointers.
  - Step (6) of the previous example profits from an inverted index on country.code.

#### Pure Index-based Equi-Joins

... via intersection of index values

Consider the equijoin:  $R \bowtie_{R.A=S.B} S$  from before.

- Indexes on R.A and on S.B (both tree or both hash with same hash function).
- find matching index entries by "merge"-style processing of leaves or tiles, yields pointer pairs (*ptr<sub>A</sub>*, *ptr<sub>B</sub>*) to matching tuples,
- access the tuples, or the required output attribute values per inverted index, (if joining further an another attribute with another table *T*, the tuple pointers/ids may be sufficient)
- with tuple access:  $O(2 \cdot |Result|)$  page accesses, with inverted index none.
- $\Rightarrow$  may be better or worse than the tuple-based algorithms.
- $\Rightarrow$  if no index is present, on-the-fly temporal indexes (or inverted indexes) can be created by the database.

(the DB may keep this indexes then, and maybe maintain them, or delete them when the relation is updated)

202

## **GROUPING AND AGGREGATION**

#### **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

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

#### UNION, DIFFERENCE, INTERSECTION $\Leftrightarrow$ CARTESIAN PRODUCT

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

#### 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

204

#### 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 and S:

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

is the proportion 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 4.6** *Compute all pairs of european countries that are adjacent to the same set of seas.*

#### Example 4.7

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

206

## 4.1.3 Exercises and Examples

Solution to Exercise 4.1

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.

 $1000 \cdot (1 \text{ (load)} + 100 \text{ (tuples per page)} \cdot 500 \text{ (pages of } S)) = 50,001,000$ 

- outer loop: *S* Load each page of *S*. For all tuples, iterate over *R*'s pages.  $500 \cdot (1 \text{ (load)} + 80 \text{ (tuples per page)} \cdot 1000 \text{ (pages of } R)) = 40,000,500$ 

Solution to Exercise 4.1 (cont'd) R: 1000 pages, each of them with 100 tuples S: 500 pages, each of them with 80 tuples Cache: 100 pages 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.  $1000 \cdot (1 + 500 \text{ (pages of } S)) = 501,000$ - outer loop: S.  $500 \cdot (1 + 1000 \text{ (pages of } R)) = 505,000$ b2) maximum-pages nested loop: load as many (first 99) pages of R as possible in the cache and join with one page of S after the other. Then continue with 2nd 99 pages. - 11 times, i.e.,  $10 \cdot 99 R$  pages +  $1 \cdot 10 R$  pages through all S pages. Overall:  $1000 + 11 \cdot 500 = 6500$ . - symmetric with  $(5 \cdot 99 + 1 \cdot 5)$  pages of S: Overall:  $500 + 6 \cdot 1000 = 6500$ . - can be algorithmically optimized by on-the-fly indexes during the main-memory join

```
208
```

#### Solution to Exercise 4.1 (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.B): Assumptions:
  - every R.A matches average 4 S-tuples.
  - index over S.B already exists (if not: creating an index over S.B requires 500 accesses) and can be kept in main memory

Iterate over R, for each tuple search for the value of R.A in the index over S.B and access the tuples

 $1000 \cdot (1 \text{ (load)} + 100 \cdot 4 \text{ (access tuples)}) = 401,000.$ 

Note: the number of page accesses depends on the number of results since for every actual result there is one page access.

In case that S is ordered wrt. S.B:  $1000 \cdot (1 \text{ (load)} + 100 \cdot 1 \text{ (access tuples)}) = 101,000.$ 



R: 1000 pages, each of them with 100 tuples

- S: 500 pages, each of them with 80 tuples; Cache: 100 pages
- d) sort-merge:

different settings

- if already ordered: linear scan: 1000 + 500 = 1500
- sort relations first:  $2 \cdot 1000 + 2 \cdot 500 + 1000 + 500 = 4500$
- using an index (assume: every R.A matches average 4 S-tuples):  $1000 + 500 + 4 \cdot 1000 \cdot 100 = 401,500$

#### 210

#### Solution to Exercise 4.1 (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:

100 pages fit in memory - read one page of R after the other and distribute over 99 partitions (whenever a page of a partition is full, move it to the disk). In the average, each partition them contains about 10.1 ( $\rightarrow$  11) pages; last page of

each partition is not completely filled.

maximum 1000 (read) + 1089 (write) = 2099.

- Same for S. Get 99 partitions of average 5.1 ( $\rightarrow$  6) pages (maximum 500 + 594 = 1094 accesses).
- now there are 99 corresponding pairs of partitions (one from *R*, one from *S*).
   Join each pair: read 11 + 6 pages per partition and process them.
   (about 99 · (11 + 6) = 1683 accesses).
- Overall: 4876 accesses.



```
212
```

#### Solution to Exercise 4.6 (cont'd)

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

 $= (seas(C_1) - seas(C_2)) \cup (seas(C_2) - seas(C_1)) = \emptyset$ 

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

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

```
SELECT ...
FROM (SELECT european countries) as C1,
(SELECT european countries) as C2
WHERE \sigma[seas(C_1, C_2)]
```

#### Solution to Exercise 4.7

- $\pi[abbrev](\sigma[all\_continents(org)])(organization)$  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  $\pi[name](continent) - \pi[enc.continent]((\sigma[organization = org]isMember) \bowtie encompasses)$ 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))

214

# Chapter 5 Relational Databases and SQL: Further Issues

- Data Definition Language (DDL):
   schema generation
- Data Manipulation Language (DML):
  - queries
  - insertions, deletions, modifications
- Database behavior?
# 5.1 Database Schema

The database schema is the complete model of the structure of the application domain (here: relational schema):

- · relations
  - names of attributes
  - domains of attributes
  - keys
- additional constraints
  - value constraints
  - referential integrity constraints
- storage issues of the physical schema: indexes, clustering etc. also belong to the schema

216

# 5.1.1 Schema Generation in SQL

#### **Definition of Tables**

Basic form: attribute names and domains

```
CREATE TABLE 
(<col> <datatype>,
:
<col> <datatype>)
```

domains: NUMBER, CHAR(n), VARCHAR2(n), DATE ...

```
CREATE TABLE City
```

(	Name	VARCHAR2(35),
	Country	VARCHAR2(4),
	Province	VARCHAR2(32),
	Population	NUMBER,
	Latitude	NUMBER,
	Longitude	NUMBER );

#### Integrity constraints

Simple constraints on individual attributes are given with the attribute definitions as "column constraints":

- · domain definitions are already integrity constraints
- further constraints on individual attribute values more detailed range restrictions: City: CHECK (population ≥ 0) or CHECK (longitude BETWEEN -180 AND 180) Note: every CHECK constraint can only use a single column.
- NULL values allowed? : Country: name NOT NULL
- Definition of key/uniqueness constraints: Country: code PRIMARY KEY or name UNIQUE

```
218
```

#### Integrity constraints (Cont'd)

Multi-attribute constraints are given separately as "table constraints":

```
CREATE TABLE
```

```
(<column definitions>,
```

```
<table-constraint>, ... ,<table-constraint>)
```

- table-constraints have a name;
- · must state which columns are concerned;
- e.g. multi-column keys and foreign keys.

#### CREATE TABLE City

```
( Name VARCHAR2(35),
```

```
Country VARCHAR2(4),
```

```
Province VARCHAR2(32),
```

```
Population NUMBER CONSTRAINT CityPop CHECK (Population >= 0),
Latitude NUMBER CONSTRAINT CityLat CHECK (Latitude BETWEEN -90 AND 90),
Longitude NUMBER CONSTRAINT CityLong CHECK (Longitude BETWEEN -180 AND 180),
CONSTRAINT CityKey PRIMARY KEY (Name, Country, Province));
```

... for details see "Practical Training SQL".

Integrity constraints (Cont'd)

· up to now: only intra-table constraints

#### General Assertions

- inter-table constraints
  e.g., "sum of inhabitants of provinces equals the population of the country",
  "sum of inhabitants of all cities of a country must be smaller the than population of the country"
- SQL standard: CREATE ASSERTION
- not supported by most systems
- · other solution: later

220

# 5.1.2 Referential Integrity Constraints

- important part of the schema; especially for tables corresponding to relationship types;
- relate foreign keys with their corresponding primary keys:



encompasses (cf. Slide 47)						
Country	Continent	Percent				
VARCHAR(4)	VARCHAR(20)	0) NUMBER				
R	Europe	20				
R	Asia	80				
D	Europe	100				
	•••					

 $encompasses.country \rightarrow country.code \quad and \\ encompasses.continent \rightarrow continent.name$ 

Tables corresponding to entity types have foreign keys that correspond to 1:n relationships:

city.country  $\rightarrow$  country.code and country.(capital,province,code)  $\rightarrow$  city.(name,province,country)

```
Referential Integrity Constraints: SQL Syntax

    as column constraints (only single-column foreign keys):

   <column-name> <datatype> REFERENCES (<column>)

    as table constraints (also compound foreign keys):

   [CONSTRAINT <name>] FOREIGN KEY (<column-list>)
                        REFERENCES (<column-list>)
CREATE TABLE encompasses
                 VARCHAR2(4) REFERENCES Country(Code),
  (Country
   Continent
                 VARCHAR2(12) REFERENCES Continent(Name),
                 NUMBER CHECK (0 < percent <= 100),
   percent
   PRIMARY KEY (Country, Continent));
CREATE TABLE City
 (Name VARCHAR2(35),
   Country VARCHAR2(4) REFERENCES Country(Code),
   Province VARCHAR2(32),
   Population NUMBER ..., Latitude NUMBER ..., Longitude NUMBER ...,
   CONSTRAINT CityKey PRIMARY KEY (Name, Country, Province),
   FOREIGN KEY (Country, Province) REFERENCES Province (Country, Name) );
```

222

# 5.1.3 Virtual Tables: Views

Views are tables that are not materialized, but *defined* by a query against the database:

```
CREATE VIEW <name> AS <query>
CREATE OR REPLACE VIEW symm_borders AS
SELECT * FROM borders
UNION
SELECT Country2, Country1, Length FROM borders;
SELECT country2
```

FROM symm\_borders WHERE country1='D';

- classical views: the content of a view is always computed when it is queried.
- Materialized Views: view is materialized and automatically maintained
   → view maintenance problem: when a base table changes, what modifications have to be applied to which views?

# 5.2 SQL: Data Manipulation Language

... everything is based on the structure of the SELECT-FROM-WHERE clause:

· Deletions:

DELETE FROM WHERE ...

- specifies in which table to delete,
- where-clause can contain arbitrary subqueries to other tables
- Updates:

- value can be a subquery (also a correlated one)
- Insertions:

```
INSERT INTO  VALUES (<const_1>, ..., <const_n>)
```

```
INSERT INTO  (SELECT ... FROM ... WHERE ...)
```

– where the < const $_i>$  are constants (strings, numbers, dates, ...).

224

# 5.3 SQL: The DATE Datatype ... and Customization

- · many applications in business and administration use dates
- computations on dates (e.g., "last of the third month after ...", "number of days between")
- $\Rightarrow$  SQL provides comprehensive datatypes DATE, TIME, TIMESTAMP
  - A More General View I: Datatypes

DATE etc. are just some (important and typical) examples of built-in datatypes

- specific operators (and behavior, cf. the XMLTYPE datatype in the SQLX standard)
- · handled via one or more lexical representations as strings

# A MORE GENERAL VIEW II: INTERNATIONALIZATION AND CUSTOMIZATION

Database systems are used anywhere in the world (like most software), *and* their contents is exchanged all over the world

- people use different languages (e.g. for error messages!)
- people use different representations
  - even for numbers: 3,1415 vs. 1.000.000 (german), 3.14 vs. 1,000,000 (anywhere else)
  - for dates: '31.12.2007', '12/31/2007' or '12-31-2007' (USA), '01-JAN-2003' etc., '01
     Janeiro 2003' even language dependent.

226

# SQL: INTERNATIONALIZATION AND CUSTOMIZATION

This issue is handled syntactically differently (but using the same idea) between different products.

#### Oracle: Natural Language Support

NLS\_LANG (language and localization issues in general), NLS\_NUMERIC\_CHARACTERS (decimal point/dezimalkomma) and NLS\_DATE\_FORMAT (date format), NLS\_SORT (sorting order)

- ALTER SESSION SET NLS\_LANGUAGE = 'Language Territory.CharacterSet'; Language: error messages, etc, Territory: more detailed formats (America/Canada/UK) including default for decimal point and date format. ALTER SESSION SET NLS\_LANGUAGE = 'portuguese'
- ALTER SESSION SET NLS\_NUMERIC\_CHARACTERS =',.'; (german style), ALTER SESSION SET NLS\_NUMERIC\_CHARACTERS ='.,'; (english style),
- ALTER SESSION SET NLS\_DATE\_FORMAT = '*string-pattern*', e.g. 'DD.MM.YYYY', 'DD-MON-YY', 'DD hh:mm:ss'

#### SQL: Internationalization and Customization

Then, e.g., INSERT INTO Politics VALUES('D', '18.01.1871', 'federal republic')

is correctly interpreted. In the output, DATE values are always represented in the currently specified format.

- $\Rightarrow$  SQL provides comprehensive datatypes DATE, TIME, TIMESTAMP
  - semantics: year/month/date/hour/minute/second timestamp: additionally fractions of seconds as decimal (Oracle: only DATE and TIMESTAMP) built-in calendar knows about length of months, leap years etc.
  - operators on date and time:
    - date + days
    - MONTHS\_BETWEEN(date1, date2), ADD\_MONTHS(date, n), LAST\_DAY(date)
    - SYSDATE

#### to\_char(*string,pattern*) and to\_date(*string,pattern*) functions

```
SELECT to_char(independence,'MM/DD/YYYY') from Politics; -- 01/18/1871
SELECT to_char(independence,'DAY') from Politics; -- wednesday
SELECT to_date('25-FEB-2012','DD-MON-YYYY')+5 from dual; -- 01-MAR-12
```

```
228
```

#### The DATE Datatype: Example

CREATE TABLE Politics				
( Country VARCHAR2(4),				
Independence DATE,				
Government VARCHAR2(120));				
ALTER SESSION SET NLS_DATE_FORMAT = 'DD MM YYYY';				
INSERT INTO politics VALUES				
('B','04 10 1830','constitutional monarchy');				

All countries that have been founded between 1200 und 1600:

SELECT Country, Independence FROM Politics WHERE Independence BETWEEN '01 01 1200' AND '31 12 1599' ORDER BY Independence;

Country	Independence
THA	01 01 1238
MC	01 01 1419
Е	01 01 1492
NL	01 01 1579

# 5.4 Beyond Relational Completeness

- The Relational Algebra and SQL are only *relationally complete*.
- can e.g. not compute the transitive closure of a relation
- applications require a more complex behavior:
  - SQL as the "core query language"
  - with something around it ...

230

# MAKING SQL TURING-COMPLETE

• embedded SQL in C/Pascal:

EXEC SQL SELECT ... FROM ... WHERE ...

embedded into Java: JDBC (Java Database Connectivity)

- SQL-92: Procedural Extensions to SQL:
  - CREATE procedures and functions as compiled things *inside* the database
  - standardized concepts, but product-specific syntax
  - basic programming constructs of a "typical" Turing-complete language:
     Variables, BEGIN ... END, IF ... THEN ... ELSIF ..., WHILE ... LOOP ..., FOR ... LOOP
  - SQL can be used inside PL/SQL statements

# "IMPEDANCE MISMATCH" BETWEEN DB AND PROGRAMMING LANGUAGES

```
(cf. Slide 3)
```

Set-oriented (relations) vs. value-oriented (variables)

• how to handle the result of a query in C/Pascal/Java?

Iterators (common programming pattern for all kinds of collections)

- explicit:
  - new/init(<query>)/open()
  - first(), next(), isempty()
  - fetch() (into a record/tuple variable)
- implicit (PL/SQL's "Cursor FOR LOOP"):

```
FOR <\!\!\text{record-variable}\!> IN <\!\!\text{query}\!>
```

LOOP

```
do something with <record-variable>
```

```
END LOOP;
```

... for details see "Practical Training SQL".

232

# 5.5 Integrity Maintenance

- if a tuple is changed/inserted/deleted it is immediately checked whether all constraints in the current database state are satisfied afterwards.
   Otherwise the operation is rejected.
- if a constraint is defined/enabled, it is immediately checked whether it is satisfied by the current database state.
   Otherwise the operation is rejected.

Any further possibilities?

Integrity Maintenance (Cont'd): referential integrity

Consider again country - organization - is member:

 $is Member. organization \rightarrow organization. abbreviation \\ is Member. country \rightarrow country. code$ 

- · deletion of a membership entry: no problem
- deletion of a country: any membership entries for it are now "dangling"
- $\Rightarrow$  remove them!

## **Referential Actions**

FOREIGN KEY isMember(country) REFERENCES country(code) ON DELETE CASCADE

- ON DELETE CASCADE: delete referencing tuple
- ON DELETE RESTRICT: referenced tuple cannot be deleted
- ON DELETE NO ACTION: referenced tuple can be deleted if the same transaction also deletes the referencing tuple
- ON DELETE SET NULL: foreign key of referencing tuple is set to NULL
- ON DELETE SET DEFAULT: foreign key of referencing tuple is set to a default value
- same for ON UPDATE

#### 234

Referential Actions: a simple example					
	(	Country			
Name	Code	Capital		Province	
Germany	p	Berlin		Berlin	
United States	USA	Washington	Dis	str. Columbia	
		./.			
CASCADE NO ACTION				1.	DELETE FROM City WHERE Name='Berlin';
	City	1		2.	DELETE FROM Country
Name	Country	Province			WHERE Name='Germany';
Berlin Washington	D USA	Berlin Distr. Columbia		3.	UPDATE Country SET code='DE'
					WHERE code='D';

	Co	untry					
Name	Code	Capital	Province				
Germany	ρ	Berlin	Berlin				
United States	US	Washington	Distr.Col.	CA	SCADE	Province	9
				]	Name	Country	Capital
SET NULL			CASCADE		Berlin	D	Berlin
City					Distr.Col.	US	Washingto
Name	Country	Province					
Berlin	D	Berlin					
Washington USA Distr.Col.			DEL	ETE FROM C	ountry		
				W	HERE Code=	'D'	
ambiguous semantics!							

236

... active behavior/reaction on events!

# 5.6 Active Databases/Triggers

- reacting on an event
  - external event/signal
  - internal event: modification/insertion/deletion
  - internal event: time
- if a condition is satisfied
- then do something/execute an action

ECA: Event-Condition-Action rules

#### **ECA-Rules**

Consider database updates only: one or more tuples of a table are changed.

- Granularity:
  - execute action once for "all updates together" (e.g., afterwards, update a sum)
  - execute action for each changed tuple (e.g. cascading update)
- Timepoint:
  - after execution of original update
  - before execution of original update
  - instead of original update
- Actions:
  - can read the before- and after value of the updated tuple
  - read and write other tables

238

#### Triggers

The SQL standard provides "Triggers" for implementation of ECA rules:

CREATE TRIGGER

- specify event: ON {DELETE | UPDATE | INSERT} OF <pl/sql-block>
- specify condition: WHEN <condition>
- specify granularity: FOR EACH STATEMENT | ROW
- specify action by pl/sql-block.

Actions are programmed using the above-mentioned procedural extensions to SQL.

#### Applications

- implementation of application-specific business rules,
- integrity maintenance,
- monitoring of assertions.
- ... for details see "Practical Training SQL".

# Chapter 6 Running a Database: Safety and Correctness Issues

- Transactions
- · Safety against failure

Not discussed here:

· Access control, Authentication

240

# 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 (complete! cf. atomicity) 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, *transactions* are *logically atomic*, but several transactions may be executed in an *interleaved* manner (see below).

242

# 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.

## Atomicity

A transaction is *logically atomic* – even when executed interleaving with others

- all-or-nothing,
- potential rollback at the end,
- ⇒ requires isolation other transactions must not use uncommitted written values (or also rolled back)
- $\Rightarrow$  rollback based on logging (see Slide 312 ff.).

Consistency

- Concept: check conditions only at the end of a transaction (COMMIT)
- · Default in DB systems: Check after each atomic action
- Optional: declare CONSTRAINTs with DEFERRABLE INITIALLY DEFERRED to postpone checks.

244

# 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

# **NESTED TRANSACTIONS [OPTIONAL]**

- internal (hierarchical) structuring of a transaction into subtransactions
- subtransactions can be executed serially, synchronous parallel, or asynchronous parallel
- transaction satisfies ACID, subtransactions only A&I.

#### Properties of Subtransactions

- atomicity
- consistency: not required only for the root transaction
- isolation: required for rollback
- durability: not possible, since rollback of a superordinate (sub)transaction required also to rollback "committed" subtransactions

246

## Properties of Subtransactions (Cont'd)

- Commit: the local commit of a subtransaction makes its effects accessible only for its superordinate transaction.
- root transaction commits if all immediate subtransactions commit.
- rollback: if some (sub)transaction is rolled back, all its subtransactions are rolled back recursively (even when they committed locally)
- visibility: all updates of a subtransaction become visible to its superordinate transaction when it commits.
   All objects that are kept by a transaction are accessible for its subtransactions.
- Effects are not visible for sibling transactions.
- above: "closed nested transactions"
- weaker visibility/isolation requirements: "open nested transactions" require more complex rollback mechanisms

# 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 are 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.

 $\Rightarrow$  enforce virtual isolation

248

# **TYPICAL ERROR SITUATIONS**

For multiuser aspects, consider a scenario where a high number of short and long transactions has to be processed:

#### (Online) Banking

A bank maintains branches at several cities; at each city multiple customers have accounts. Customers are doing money transfers, cash withdrawals at ATMs (german: Geldautomaten); and the bank computes the yearly interest rate (german: Zinsen) always on January 1st.

Consider the following relations:

- Account: Name, City, Amount
- Branch: City, Total where "Total" contains the sum of all accounts at that place.

upuale	
money transfer $A \rightarrow B$	B taking cash at the ATM
SELECT amount INTO a	
FROM Accounts WHERE name = 'Alice'	
a := a - 100;	
UPDATE Accounts	
SET amount = a WHERE name = 'Alice'	
SELECT amount INTO b	
FROM Accounts WHERE name = 'Bob'	
	SELECT amount INTO c
	FROM Accounts WHERE name = 'Bob'
b := b + 100;	
	c := c - 200;
UPDATE Accounts	
SET amount = b WHERE name = 'Bob'	
	UPDATE Accounts
	SET amount = c WHERE name = 'Bob'
and a second second from (fine to see all to ) is to set	

• note: such problems can usually occur when values are calculated using earlier ones, not when the database does mainly store-and-read (like Mondial)

money transfer $A \rightarrow D$ fails	A taking cash at the ATM
SELECT amount INTO a FROM Accounts WHERE name = 'Alice'	
a := a - 100;	
UPDATE Accounts SET amount = a WHERE name = 'Alice' SELECT amount INTO d FROM Accounts WHERE name = 'Dave' search and wait	
	<ul> <li>SELECT amount INTO c</li> <li>FROM Accounts WHERE name = 'Alice'</li> <li>c := c - 200;</li> <li>UPDATE Accounts</li> <li>SET amount = c WHERE name = 'Alice'</li> </ul>
Dave does not have an account here! ABORT (i.e., ROLLBACK)	



#### Phantom

<ul> <li>sum of account balances equals stored total for each branch?</li> </ul>					
Check sum of account balances by branch	Insert new account				
SELECT SUM(amount) INTO sum					
FROM Account WHERE city = 'Frankfurt'					
	INSERT INTO Accounts (name, city, amount) VALUES ('Dave', 'Frankfurt', 1000)				
	UPDATE Branches SET total = total + 1000 WHERE city = 'Frankfurt'				
SELECT total INTO x FROM Branches WHERE city = 'Frankfurt'					
IF $x \neq$ sum THEN <error handling=""></error>					
<ul> <li>similar to non-repeatable read</li> </ul>					

# 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*<sub>1</sub> = *RA*; *A*:=*A*-10; *WA*; *RB*; *B*:=*B*+10; *WB*, *T*<sub>2</sub> = *RA*; *A*:=*A*-20; *WA*; *RB*; *B*:=*B*+20; *WB* 

Some schedules (without computation steps):

 $S_{1} = R_{1}A \ W_{1}A \ R_{1}B \ W_{1}B \ R_{2}A \ W_{2}A \ R_{2}B \ W_{2}B$ (serial)  $S_{2} = R_{1}A \ R_{2}A \ W_{1}A \ W_{2}A \ R_{1}B \ R_{2}B \ W_{1}B \ W_{2}B$   $S_{3} = R_{1}A \ W_{1}A \ R_{2}A \ W_{2}A \ R_{1}B \ W_{1}B \ R_{2}B \ W_{2}B$   $S_{4} = R_{1}A \ W_{1}A \ R_{2}A \ W_{2}A \ R_{2}B \ W_{2}B \ R_{1}B \ W_{1}B$  $S_{5} = R_{2}A \ W_{2}A \ R_{2}B \ W_{2}B \ R_{1}A \ W_{1}A \ R_{1}B \ W_{1}B$ (serial)

... which of them are "good"?

(for n transactions, there is an exponential number of candidates)

254

# SERIALIZABILITY CRITERION FOR PARALLEL TRANSACTIONS

- "Isolation" requirement:
   A transaction must 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<sub>1</sub>: RA; A:=A-10; WA; RB; B:=B+10; WB T<sub>2</sub>: 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	
$S_1$ :	$S_1$ : serial		$S_2$ : not serializable		$S_3$ : serializable		alizable?
A=-2	0, B=40	A=-10	, <i>B=30</i>	A=-20	),B=40	A=-20	),B=40
A+	B=20	A+E	3=20	A+E	3=20	A+E	3=20

256

Problem: what means "equivalence" in this context?

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

#### Example 6.3

Consider again Example 6.2 for A=B=10;  $T'_1: RA; A:=A^{*1.05}; WA; RB; B:=B^{*1.05}; WB$  (Yearly Interest Rate) and  $T_2: RA; A:=A-10; WA; RB; B:=B+10; WB$  (Money Transfer). Consider  $S_1, S_5 := T_2T_1, S_3$ , and  $S_4$ .

# 6.4.1 Formalization of the Semantics of Transactions

- · How to show that for all possible circumstances, a schedule is serializable?
- Theory & algorithms depend only on the READ and WRITE actions, not on the semantics of the computations in-between.

(this would require theorem-proving instead of symbolic algorithms)

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

• take a logic-based framework!

258

# ASIDE: BASIC NOTIONS OF FIRST-ORDER PREDICATE LOGIC

(you probably have learnt this in "Discrete Mathematics" or in "Formal Systems")

- An first-order signature Σ contains function symbols and predicate symbols, each of them with a given arity (function symbols with arity 0 are constants).
- The set of **ground terms** over  $\Sigma$  is built inductively over the function symbols: for  $f \in \Sigma$  with arity n and terms  $t_1, \ldots, t_n$ , also  $f(t_1, \ldots, t_n)$  is a term.
- A first-order structure S = (D, I) over a signature  $\Sigma$  consists of a nonempty set D (domain) and an interpretation I of the signature symbols over 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  $I(p) \subseteq \mathcal{D}^n$ .

# LOGIC FORMALIZATION OF THE SEMANTICS OF TRANSACTIONS

- Let  $\ensuremath{\mathcal{D}}$  denote the domain of the database objects.
- Consider a transaction *T*, with a write action *WX* where  $RY_1, \ldots, RY_k$   $k \ge 0$  are the read actions that are executed by *T* before *WX*.
- The value written to X by WX is denoted by

$$f_{T,X}(Y_1,\ldots,Y_k)$$

where

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

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

- the functions  $f_{T,X}$  abstract the calculation of the value of X that is then written by WX in T,
- their actual interpretation is given by the computation of the transaction.

260

# **APPLICATION TO TRANSACTIONS AND SCHEDULES**

- for every transaction and every schedule, the final values (call them  $a_{\infty}, b_{\infty}, ...$ ) can be expressed in terms of the  $t_{T,X}$  of the contributing transactions,
- the constants  $a_0$ ,  $b_0$  are interpreted by initial values,
- the actual interpretation of the functions is given by the transaction.

Consider the single transaction runs:

 $T_1$ : RA $T_1'$ : RAread  $a_0$ read  $a_0$ A := A - 10A := A \* 1.05write  $f_{T'_1,A}(a_0)$ write  $f_{T_1,A}(a_0)$ WA WA RBRBread  $b_0$ read  $b_0$ B := B + 10B := B \* 1.05write  $f_{T_1,B}(a_0, b_0)$ write  $f_{T'_{1},B}(a_{0},b_{0})$ WBWB $a_{\infty} = f_{T_1,A}(a_0),$  $\begin{array}{l} a_{\infty} = f_{T_{1}',A}(a_{0})\text{,} \\ b_{\infty} = f_{T_{1}',B}(a_{0},b_{0}) \end{array}$  $b_{\infty} = f_{T_1,B}(a_0, b_0)$ 

both induce the same final term structure. The interpretations differ:  $T_1: f_{T_1,A}(A) = A - 10, f_{T_1,B}(A, B) = B + 10,$  $T'_1: f_{T'_1,A}(A) = A * 1.05, f_{T'_1,B}(A, B) = B * 1.05.$ 

#### Application to Single Transactions

• for given transactions (i.e. a given interpretation of  $f_{T,X}$ ), properties of the final values can formally be proven, e.g.,

 $T_1: a_\infty + b_\infty = a_0 + b_0,$ 

 $T_1': a_{\infty} + b_{\infty} = (a_0 + b_0) * 1.05$ 

 later/Exercise: intra-transactional optimization by interchanging non-conflicting operations of a transaction.

262

#### Application to Schedules

#### Example 6.4

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  $a_0, b_0$ .

	S	chedule $T_1 T_2$ (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	$f_{T_1,B}(f_{T_2,A}(a_0), f_{T_2,B}(a_0,b_0))$

- for a given interpretation, the evaluation of the terms yields the final values,
- the terms themselves additionally encode the data flow through the schedule.

# EQUIVALENCE OF SCHEDULES

## **Definition 6.2**

Two schedules S, S' (of the same set of transactions) are equivalent, if for every initial state, corresponding atomic actions read/write the same values in S and S'.

## Corollary 6.1

For two equivalent schedules S and S' executed on the same initial state, S and S' generate the same final states.

Proof: consider the last write actions for each data item.

- according to the Definition 6.2, equivalence can only be checked by investigating both schedules step-by-step.
- In the above formalization, this is encoded into the (final) terms: the execution of a schedule is traced symbolically
   ("Herbrand interpretation" every term is interpreted "as itself").

264

# EQUIVALENCE OF SCHEDULES

#### Exercise 6.1

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.

## Example 6.5 (Solution of Exercise 6.1)

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

and the schedules  $S_2$  and  $S_4$ . Let the initial state again be given by values  $a_0, b_0$ .

Sch	nedule $S_2$	Schedule $S_4$		
RA	$a_0$	RA	$a_0$	
RA	$a_0$	WA	$f_{T_1,A}(a_0)$	
WA	$f_{T_1,A}(a_0)$	RA	$f_{T_1,A}(a_0)$	
WA	$f_{T_2,A}(a_0)$	WA	$f_{T_2,A}(f_{T_1,A}(a_0))$	
RB	$b_0$	RB	$b_0$	
RB	$b_0$	$W\!B$	$f_{T_2,B}(f_{T_1,A}(a_0),b_0)$	
WB	$f_{T_1,B}(a_0,b_0)$	RB	$f_{T_2,B}(f_{T_1,A}(a_0),b_0)$	
WB	$f_{T_2,B}(a_0,b_0)$	$W\!B$	$f_{T_1,B}(a_0, f_{T_2,B}(f_{T_1,A}(a_0), b_0))$	

For  $S_3$ , the terms are the same for every R/W as for  $S_1$ . For  $S_4$ , the blue-red-blue "shows" that there can be no serial schedule that generates the same terms.

266

# **A STEP TOWARDS MORE ABSTRACTION**

Summary and conclusions:

- the *f*s are abstractions for the actual functions/computations of the transactions,
- we are actually not interested at all, what the *f*s are,
- but (mainly) in the term structure and the data flow (indicated by the colors above),
- the "history" of a data item is described by the *f*-terms.
- $\Rightarrow\,$  find another way to represent how information flows and "who reads and writes what values".

# 6.4.2 Theoretical Investigations

Consider a schedule *S* together with two additional distinguished transactions  $T_0, T_\infty$ :  $T_0$  generates the initial state, and  $T_\infty$  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.
- $T_{\infty}$  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 6.2

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  $\hat{S'}$ .

Check all these terms for an exponential number for candidates?

268

# **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 Ŝ = ... R<sub>i</sub>B ... W<sub>i</sub>A ..., then R<sub>i</sub>B → W<sub>i</sub>A ∈ E,
  (i.e., T<sub>i</sub> reads B (and possible uses it) and then writes a value A)
- if  $\hat{S} = \ldots W_i A \ldots R_j A \ldots$ , then  $W_i A \to R_j A \in E$ , if there is no write action to A between  $W_i A$  and  $R_j A$  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 6.6



270

## ... and now to the

Proof [Optional]

Each transaction *T* with  $n, n \ge 1$  write actions on  $A_1, \ldots, A_n$  induces a set  $F_T = \{f_{T,A_1}, \ldots, f_{T,A_n}\}$  of function symbols that are used for representing the computations associated with the write actions.

Given a domain  $\mathcal{D}$ , every transaction also induces an interpretation

 $\mathcal{S}_{\mathcal{T}} = (\mathcal{D}, I_{F_T})$  such that each  $I(f_{T,A_i})$  is a mapping  $\mathcal{D}^{k_i} \to \mathcal{D}$ .

Analogously, the interpretation of a set  $T_1, \ldots, T_m$  of transactions has the form  $S = (D, F_{T_1} \cup \ldots \cup F_{T_m}).$ 

Assume an action a of a schedule S. If a is a write action, then  $a_S(I)$  is the value that is written by a in S under the interpretation I. If a is a read action, then  $a_S(I)$  is the value that is read by a.

For a node *a* of the D-graph DG(S), the **restriction** of DG(S) to *a* and its predecessors is denoted by  $pred_S(a)$  – (this is the portion of the graph consisting of all actions that contribute to the value that is read/written by *a*).

## Proof (Cont'd)

" $\Leftarrow$ ": We show that for all actions a in S,

$$pred_S(a) = pred_{S'}(a) \Rightarrow a_S(I) = a_{S'}(I)$$

for arbitrary interpretations I by induction over the number of nodes in  $pred_S(a)$ .

Assume that a is an action in a transaction T to a database object x.

- $pred_S(a)$  contains a single node. Then, *a* is this node.
  - *a* cannot be a read action, as any read action RA would have at least a write action  $W_0$  in  $T_0$  as predecessor.
  - if *a* is a write action on *A*,  $f_{T,A}$  is a constant function (depending on no input/original values) and thus,  $a_S(I) = a_{S'}(I)$  for all *I*.
- $pred_S(a)$  contains more than a single node. Because of  $pred_S(a) = pred_{S'}(a)$ , *a* has the same predecessors  $b_1, \ldots, b_k$  in both graphs. By induction hypothesis, for each of them,  $b_S(I) = b_{S'}(I)$ .
  - if *a* is a read action, the conclusion  $a_S(I) = a_{S'}(I)$  is again trivial.
  - if a is a write action on A,

$$a_{S}(I) = f_{T,A}(b_{1S}(I), \dots, b_{kS}(I)) = f_{T,A}(b_{1S'}(I), \dots, b_{kS'}(I)) = a_{S'}(I).$$

Thus, in both sequences, the same values are read and written.

272

#### Proof (Cont'd)

"⇒":

Since S and S' are assumed to be equivalent, all transactions in S and S' read the same values for arbitrary interpretations I.

Consider the (Herbrand-) [that means, using uninterpreted ground terms] interpretation H to the transactions in S:

•  $\mathcal{D} = \{ f_{T_0,A_1}, f_{T_0,A_2}, \dots, f_{T_1,A_1}(\dots, f_{T_0,A_1}, \dots), \dots, \}$ 

 $f_{T_2,A_1}(\ldots,f_{T_0,A_1},\ldots,f_{T_1,A_1}(\ldots,f_{T_0,A_1},\ldots),\ldots),\ldots\}$ 

is the set of (ground) terms built over the symbols that are assigned to the write actions.

•  $f_{T,A}: \mathcal{D}^k \to \mathcal{D}$ : applying  $f_{T,A}$  to values  $v_1, \ldots, v_k$  yields the term  $f_{T,A}(v_1, \ldots, v_k)$ .

For every action a in S and S',  $a_S(H)$  and  $a_{S'}(H)$  encode  $pred_S(v)$  and  $pred_{S'}(v)$ , resp.

(e.g., if for a write action  $a = W_1B$ ,  $a_S(H) = f_{T_1,B}(a_0, f_{T_2,B}(f_{T_1,A}(a_0), b_0))$ , then it has a predecessor  $R_1A$  that read  $a_0$ , and a predecessor  $R_1B$  that read the value  $f_{T_2,B}(f_{T_1,A}(a_0), b_0)$  written by  $W_2B$  that in course had (i) a predecessor  $R_2A$  that read value  $f_{T_1,A}(a_0)$  written by  $W_1A$  that in turn had a predecessor  $R_1A$  that read  $a_0$ , and (ii) a predecessor  $R_2B$  that read  $b_0$ ).

Thus, since  $v_S(H) = v_{S'}(H)$ , DG(S) = DG(S').

# NEXT STEP TOWARDS MORE ABSTRACTION

П

• we are even not interested in the Dependency Graph, only in the question for which schedules there is a serial schedule with the same DG.

#### Example 6.7

Consider the DG of  $S_4$  Example 6.2 (Example/Exercise)

- intra-transaction edges are not relevant (for a given transaction they are the same in all DGs),
- edges between transactions are important (see above example),
- some other relationships between transactions are also important.
- $\Rightarrow$  they tell, what conditions an equivalent serial schedule must satisfy!
  - ... if they are satisfiable, there is an equivalent serial schedule (or several of them).

# THEORY: EQUIVALENCE CLASSES OF SCHEDULES

#### Recall from Discrete Mathematics

A binary relation  $\sim$  is an equivalence relation on a set X if it is

- reflexive:  $x \sim x$  for every  $x \in X$ ,
- symmetric:  $x \sim y \Rightarrow y \sim x$  for every  $x, y \in X$
- transitive  $x \sim y \wedge y \sim z \Rightarrow x \sim z$  for every  $x, y, z \in X$ .

#### **Equivalence Classes**

For an equivalence relation  $\sim \subseteq X \times X$ , the equivalence class [x] is defined as

 $[x] := \{ y \in X | x \sim y \}$ 

Note: two equivalence classes are either the same, or disjoint.

## Equivalence Classes of Schedules and Serializable Schedules

On the set of schedules, let  $\sim$  be defined as  $S \sim S'$  if DG(S) = DG(S'). A schedule S is serializable, if  $S \in [S']$  for a serial schedule S'.

The following properties hold:

- given n transactions, there are at most n! equivalence classes of serializable schedules,
- for two serial schedules,  $[S_1] = [S_2]$  is possible (when two or more transactions have no conflicts at all),
- there are many more equivalence classes of non-serializable schedules.

276

## Neighboring Schedules

#### **Definition 6.3**

Two schedules S, S' (of the same set of transactions) are neighbors if S' can be obtained from S by exchanging a single pair of atomic actions  $a_1, a_2$ .

Note:

- for a given set of transactions T,  $a_1$ ,  $a_2$  above must belong to different transactions to obtain a valid schedule of T.
- Aside: if exchanging actions of *the same* transaction, the approach is applicable to intra-transaction optimization:

Actions in a transaction can be exchanged if the D-Graph is not effected (e.g.,  $R_iA$  and  $R_iB$ ).

# WHEN ARE NEIGHBORING SCHEDULES EQUIVALENT?

Let  $S = S_1 a_i a_j S_2$  and  $S' = S_1 a_j a_i S_2$  be neighboring schedules.

Consider each pair of types of actions possible for  $(a_i, a_j)$ .

- obviously, actions on *different* data items can be exchanged without effecting the D-Graph.
- **RR:**  $R_iA, R_jA$ : no change.
- **WR:**  $W_iA, R_jA$ : WR is an edge in the D-Graph, exchanging the actions removes this edge and adds an edge from the preceding  $W_jA$  to  $R_jA$ .
- **RW:**  $R_iA, W_jA$ : symmetric. RW represents a "no-edge" in the DG.
- **WW:**  $W_iA, W_jA$ : For the next  $R_kA$  (if no  $W_\ell$  is in-between [this condition will become relevant later note also that  $T_\infty$  is needed here]), there is an edge in the D-Graph  $W_jA \to R_kA$ ; after exchanging, there is an edge  $W_iA \to R_kA$ .

In the RW/WR/WW cases the D-Graph is different from before,  $S \not\sim S'$ .

The respective pairs of actions  $a_i, a_j$  determine a constraint (that distinguishes *S* from *S'* and *S* from *S'*) on the equivalent serial schedule that  $T_i$  must be executed before  $T_j$ .

278

# **CONFLICT GRAPH: IDEA**

Every schedule can be characterized wrt. the equivalence class it belongs to by these "borders" between their member sets.

- the constraints state a topological order on the set T of transactions,
- if the graph is cyclic, then the set of constraints is not satisfiable (= there is no equivalent serial schedule).

Note that [S] then also exists, but does (usually; cf. later) not contain any serial schedule; only under certain conditions, there may be an equivalent serial schedule.

• if the graph does not contain a cycle, the constraints can be interpreted as a topological order that characterizes the equivalence class.

# **CONFLICT GRAPH: DEFINITION**

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 given as follows ( $i \neq j$ ):

- if  $S = \ldots W_i A \ldots R_j A \ldots$  then  $T_i \to T_j \in E$ , if there is no write action to A between  $W_i A$  and  $R_i A$  in S (WR-conflict).
- if  $S = \ldots W_i A \ldots W_j A \ldots$  then  $T_i \to T_j \in E$ , if there is no write action to A between  $W_i A$  and  $W_j A$  in S (WW-conflict).
- if  $S = \ldots R_i A \ldots W_j A \ldots$  then  $T_i \to T_j \in E$ , if there is no write action to A between  $R_i A$  and  $W_j A$  in S (**RW-conflict**).

#### Theorem 6.2

If the conflict graph CG(S) of a schedule S is cycle-free, then S is serializable.

#### 280

#### Conflict Graph Theorem: Proof

Assumed: Since CG(S) is cycle-free.

Interpret CG(S) as a topological order of the nodes (i.e., of the transactions). Short: let S' a serial schedule according to this ordering. Then, DG(S) = DG(S') and [S] = [S'].

Long:

Let  $CG^*(S)$  denote the transitive closure of CG(S).

For any serial  $S' = T_{i_1} \dots T_{i_n}$  over  $T_1, \dots, T_n$  (i.e.,  $\{i_1, \dots, i_n\} = \{1, \dots, n\}$ ), let

 $\leq_{S'} := \{(n,m) | T_n \text{ occurs before } T_m \text{ in } S'\}$ 

 $S' \sim S \Leftrightarrow CG^*(S) \subseteq \leq_{S'}$  (i.e., if the orderings are consistent).

As CG(S) is noncyclic, it is a (satisfiable) topological ordering and such an S' exists.

#### Remarks

Note that the ordering given by CG(S) may be incomplete, i.e. there can be serial  $S' \neq S''$  both in the same equivalence class: [S] = [S'] = [S''].

# **CONFLICT GRAPHS**

#### Example 6.8

Consider the CGs of  $S_1$ ,  $S_3$ ,  $S_4$  from Example 6.2.

#### Example 6.9

Consider the schedule

 $S = R_1 A W_1 A R_2 A R_3 A R_2 B R_3 C W_3 C W_2 B.$ 

Draw the Conflict Graph, interpret it as a topological order and give all equivalent serial schedules. Draw the DG of *S* and the DGs of the equivalent serial schedules.

282

# **NEIGHBORING SCHEDULES**

• Given a serial schedule, equivalent schedules can be constructed by considering neighboring schedules:

"It is allowed to postpone action  $a_i$  of  $T_i$  and instead already process action  $a_j$  from  $T_j$ ?" (cf. Schedule  $S_3$  in Example 6.2)

- define  $\sim_1$  (one-change-equivalence) by

 $S_1 \sim_1, S_2 \ :\Leftrightarrow \ S_1$  and  $S_2$  are neighbors and  $S_1 \sim S_2$ 

- define  $\sim_n$  (*n*-change-equivalence) by

 $S_1 \sim_n, S_2 \iff$  there are  $S_1, S_2, \ldots, S_n$  such that  $S_i \sim_1 S_{i+1}$  for all  $1 \le i < n$ 

• Obviously,  $\sim \subseteq \bigcup_{n \in \mathbb{N}} \sim_n$ . (When) does equality hold?


284

# **CLOSER LOOK AT WW CONFLICTS**

Consider again Slide 278 and  $W_i A W_j A$ -conflicts:



If there is another  $W_{\ell}A$  before the next  $R_kA$ , the D-Graph is *not* changed when interchanging  $a_i$  and  $a_j$ :



- A write  $W_j A$  "cuts" the data flow from the preceding  $W_i A$ .
- WW-conflicts where the second write is never read can be ignored.
- the above fragments can only be completed to equivalent serializable schedules if  $W_{\ell}A$  and either  $W_iA$  or  $W_jA$  are blind writes.

# **CLOSER LOOK AT RW CONFLICTS**

Consider again Slide 278 and  $R_i A W_j A$ -conflicts:

$$R_i A$$
  
 $W_j A$ 

$$R_i A \overset{W_j A}{\checkmark} W_j A$$

Exchanging  $R_i A$  and  $W_j A$  leads to a dataflow. If  $W_j A$  is not put immediately before, but much earlier, the original dataflow is (locally) unchanged:

• the above fragments can only be completed to equivalent serializable schedules if  $W_jA$  is followed by a blind write in both cases (which is exactly the case as in Example 6.10).

# EQUIVALENT SCHEDULES IN PRESENCE OF BLIND WRITES

- Example 6.10 shows that in presence of two blind writes a completely different schedule can be equivalent.
- WR-conflicts represent actual data flow they must be the same in the corresponding serial schedule.
- WW-conflicts and RW-conflicts can be ignored under certain conditions (needing at least two blind writes on the corresponding data item)
- In the RW case, there is no path wrt. the "neighborship" relation from S' to S that stays inside [S] = [S'], i.e.,  $[S] \not\sim_n [S']$  for any n.
### C-Graph-Serializability

### **Definition 6.4**

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

### Theorem 6.3

If for a set  $\mathcal{T}$ , there are no "blind writes", i.e., for each  $T \in \mathcal{T}$ ,

 $T = \dots WA \dots \Longrightarrow T = \dots RA \dots WA \dots,$ 

then every schedule S over T is serializable if and only if S is C-serializable.

Proof: Exercise.

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

288

# 6.4.3 More Detailed Serializability Theory [Optional]

The C-Graph is more restrictive than necessary (cf. the above example):

For a more liberal criterion, consider only the following situation:

$$S = \ldots W_i A \ldots R_j A \ldots$$

and there is no write action on A between  $W_i$  and  $R_j$ .

- In every equivalent serial schedule,  $T_i$  precedes  $T_j$ ,
- if  $W_kA \in S$ , there is no equivalent serial schedule s.t.  $T_k$  is between  $T_i$  and  $T_j$ . But, it can be *before*  $T_i$  *or after*  $T_j$ .

### POLYGRAPH

For a schedule S, the **polygraph** P(S) is a tuple P(S) = (V, E, F), where

- 1. V is the set of transactions in S,
- 2. E is a set of edges, given by the WR-conflicts in S,
- 3. *F* is a set of pairs of edges (alternatives): for all  $i \neq j$  such that  $S = \dots W_i A \dots R_j A \dots$  and there is no write action to *A* between  $W_i A$  and  $R_j A$ , and all  $W_k A$  in *S* where  $k \neq i, k \neq j$ :

$$T_k \to T_i, T_j \to T_k) \in F_k$$

(include  $start = W_0(all)$  and  $end = R_{\infty}(all)!$ )

A graph (V, E') is **compatible** to a polygraph (V, E, F) if  $E \subseteq E'$  and E' contains for each alternative exactly one of the edges.

A polygraph P(S) = (V, E, F) is **cycle-free** if there is a cycle-free compatible graph (V, E').

### Theorem 6.4

A schedule S is serializable if and only if its polygraph is cycle-free.

Note: The test for cycle-freeness of a polygraph is NP-complete.

290

### Example 6.11

Consider again Example 6.10.

 $\begin{array}{cccc} T_1: & \mathcal{R}(\mathcal{X}) & \mathcal{W}(\mathcal{Y}) \\ \mathcal{S}: & T_2: & \mathcal{R}(\mathcal{Y}) & \mathcal{W}(\mathcal{Y}) \\ T_3: & \mathcal{R}(\mathcal{Z}) & \mathcal{W}(\mathcal{Y}) \end{array}$ 

From (2), there is the edge  $(3,2) \in E$ .

From (3), consider

- $W_3(Y)/R_2(Y)$ : For  $W_1(Y)$  ((1,3), (2,1)) has to be added to F.
- $W_0(X)/R_1(X)$ ,  $W_0(Z)/R_3(Z)$ : there is no W(X) and W(Z). Do nothing. Note that the original value of Y is never read.
- $W_2(Y)/R_{\infty}(Y)$ . For  $W_1(Y)$  and  $W_3(Y)$ , add  $((1,2), (\infty,1))$  and  $((3,2), (\infty,3))$  to F.

Since edges like (n,0) (a transaction before start) and  $(\infty, n)$  (a transaction after end) do not make sense, the only compatible graphs are

- (3,2), (1,3), (1,2), (3,2) (cycle-free), and
- (3,2), (2,1), (1,2), (3,2) (cyclic).

The first of these gives the equivalent serial schedule  $T_1T_3T_2$ .

... and now to the

Proof of Theorem 6.4

We need two Lemmata:

### Lemma 6.1

For two equivalent schedules S and S', P(S) = P(S').

Proof: Follows from equality of the D-Graphs (which are also based on WR-conflicts).

### Lemma 6.2

For a serial schedule S, P(S) is cycle-free.

**Proof:** Construct a graph *G* that contains an edge  $T_i \rightarrow T_j$  if and only if  $T_i$  is before  $T_j$  in *S*. *G* is cycle-free and compatible to P(S).

### Proof of the theorem

" $\Rightarrow$ ": follows immediately from the above lemmata.

" $\Leftarrow$ ": Consider a cycle-free graph *G* that is compatible to P(S). Let *S'* a serial schedule according to a topological sorting of *G*.

We show that S and S' are equivalent, i.e., DG(S) = DG(S').

Assume  $DG(S) \neq DG(S')$ . Thus, there are actions  $W_iA, W_kA, R_jA$  from different transactions such that

- in S,  $T_j$  reads a value of A that has been written by  $T_i$ . Thus,
  - the *E* component of P(S) contains an edge  $T_i \rightarrow T_j$ ,
  - The *F* component of P(S) contains a pair  $(T_k \to T_i, T_j \to T_k)$ .
- in S',  $T_j$  reads a value of A that has been written by  $T_k$ .

Because of compatibility, *G* contains the edge  $T_i \rightarrow T_j$ . Since *S'* is serial, it is of the form  $S' = \dots T_i \dots T_j \dots$ 

Since  $T_j$  reads A from  $T_k$  in S' (assumption),  $T_k$  is be executed later than  $T_i$ , and before  $T_j$ . Thus,  $S' = \ldots T_i \ldots T_k \ldots T_j \ldots$ 

Since G is cycle-free, there are no edges  $T_k \to T_i$  or  $T_j \to T_k$ .

Then, G cannot be compatible to P(S) (the pair in F is not satisfied).

# 6.5 Scheduling

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

- runtime-scheduling, incremental, "online-algorithm" that does not need to test an exponential number of possibilities a priori, but which runs nearly in linear time.
- at each timepoint, new transactions can "arrive" and have to be considered

**Different Types of Strategies** 

- Supervise the schedule, and with the first non-serializable action, kill the transaction (  $\rightarrow$  C-graph, timestamps)
- Avoidance strategies: avoid at all that non-serializable schedules can be created ( $\rightarrow$  Locking),
- Optimistic Strategies: keep things running even into non-serializable schedules, and check only just before committing a transaction (→ read-set/write-set).

294

### Scheduling Strategies

### Based on the conflict graph:

The scheduler maintains the conflict graph of the actions executed so far (partial schedule).

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 (*action* will never be conflict-free in this schedule) abort T and all transactions that depend on T (i.e., that have read items that have been written by T before), and remove the corresponding actions from S. Restart T later.

Note: not only the CG must be maintained, but all earlier actions that can still be part of a conflict (i.e., for each tuple, all actions backwards until (including) the most recent write).

Exercise:  $S_4$  from Example 6.2.

### Scheduling Strategies (Cont'd)

### 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. Restart T later (with new timestamp).

**Implementation:** For any action (read and write) on a data item V, the latest timestamp is recorded at V as  $Z_r(V)$  or  $Z_w(V)$ . Then, (\*) is checked as  $Z_?(T) \ge Z_?(V)$  (set "?" according to conflict matrix), and if an action is executed, then Z(V) is set to Z(T).

• Lock-based strategies: see next section.

296

### Scheduling Strategies (Cont'd)

### Optimistic Strategies:

(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.

# 6.6 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<sub>R</sub>X)
  - Read and write-privilege: WLOCK (L<sub>W</sub>X)
- · 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 6.12

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"): granted privilege (for the same object): requested privilege LOCK LOCK Ν • if read and write privilege are distinguished: RLOCK WLOCK i.e., multiple transactions reading the same ob-RLOCK Υ Ν ject are allowed. WLOCK Ν Ν



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  $T_i \rightarrow T_j$  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.

302

Note: locks alone do not yet guarantee serializability.

### Example 6.13

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):

 $S_{4L} = L_1 A R_1 A W_1 A U_1 A L_2 A R_2 A W_2 A U_1 A$  $L_2 B R_2 B W_2 B U_2 B L_1 B R_1 B W_1 B U_1 B$ 

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 6.14

Consider again Example 6.12:

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.

304

### Theorem 6.5

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

 $S = \dots (W_{1}A_{1}) \ U_{1}A_{1} \ \dots \ L_{2}A_{1} \ (R_{2}A_{1}) \ \dots$   $S = \dots (W_{2}A_{2}) \ U_{2}A_{2} \ \dots \ L_{3}A_{2}(R_{3}A_{2}) \ \dots$   $\vdots$   $S = \dots (W_{k-1}A_{k-1}) \ U_{k-1}A_{k-1} \ \dots \ L_{k}A_{k-1} \ (R_{k}A_{k-1}) \ \dots$   $S = \dots (W_{k}A_{k}) \ U_{k}A_{k} \ \dots \ L_{1}A_{k} \ (R_{1}A_{k}) \ \dots$ 

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.

 $(T_1 \text{ is then of the form } \dots UX \dots LY \dots)$ 

### Example 6.15

Consider the non-2PL transaction from Example 6.14 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) is possible that has been shown not to be serializable:

 $S = L_1 A R_1 A W_1 A U_1 A L_2 A L_2 B R_2 A W_2 A$  $R_2 B W_2 B U_2 A U_2 B L_1 B R_1 B W_1 B U_1 B$ 

306

### Properties of 2PL (Cont'd)

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

### Example 6.16

The schedule  $\boldsymbol{S}$ 

### $S = R_1 A R_2 A W_2 A R_3 B W_3 B R_1 B W_1 B$

is serializable (equivalent to  $T_3 T_1 T_2$ ), but there is no way to add LOCK/UNLOCK actions to  $T_1$  that satisfy the 2PL requirement such that S is an admissible schedule.

# STRICT 2PL

Consider Schedule  $S_3$  from Example 6.2 with 2PL-Locks:

 $S_3 = L_1 A R_1 A W_1 A L_1 B U_1 A L_2 A R_2 A W_2 A L_2 B U_2 A R_1 B W_1 B U_1 B R_2 B W_2 B U_2 B.$ 

Consider the case that  $T_1$  fails as follows:

 $L_1A R_1A W_1A L_1B U_1A L_2A R_2A W_2A L_2B U_2A R_1B \mathsf{ROLLBACK}_1$ 

- $T_2$  has already read A (dirty read) and must also be rolled back.
- Dirty reads (and cascading rollbacks) can be avoided, if the locks are only released *after* EOT ("Strict 2PL"):  $T_1 = L_1 A R_1 A W_1 A L_1 B R_1 B W_1 B EOT_1 U_1 A U_1 B$ .
- the user does not have to specify Lock/unlock at all:
  - every item is locked when used for the first time (done via the Access Manager),
  - the transaction manager unlocks all items of a transaction after EOT.



# LOCKING GRANULARITY

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

### Example 6.17

Consider again Slide 253.

 $T_1$  computes the sum of the population of all accounts in Frankfurt – reading all these tuples. Thus, at the beginning it locks all (existing) tuples.  $T_2$  adds a new account and adapts the total.

The schedule given on Slide 253 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.

310

# 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* EOT (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)

# 6.7 Safety: Error Recovery

What (more or less 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** ( $\Rightarrow$  no cascading rollback).

312

# "SIMPLE" ROLLBACK

- the transaction manager decides to rollback a running transaction,
- · requires to undo all effects of the database
- (recall that strict 2PL is assumed, which avoids dirty reads),
- requires for each transaction a list of what it did.
- these lists could be kept separately for each transaction, or altogether in a "database log" (which will prove useful also in more severe error situations)



# TRANSACTION ROLLBACK WITH A DATABASE LOG

Consider the following "money transfer" transaction  $T_1$  which additionally sends a confirmation e-mail to A.

 $T_1 = R_1 A(x), x = x - 100, W_1 A(x), R_1 B(y), y = y + 100, W_1 B(y), R_1 C(m)$ , send mail to *m*. Given  $a_0 = 1000, B_0 = 2000$ , and assume that the sending of the mail fails, the log looks as follows:

 $\dots (T_1, begin) \dots (T_1, A, 1000, 900) \dots (T_1, B, 2000, 2100) \dots$ 

Now, execution of  $T_1$  fails when sending the mail.

Scanning the log backwards for entries on  $T_1$ : set *B* back to 2000, set *A* back to 1000, stop going backwards when  $(T_1, begin)$  is reached.

(preferable: have an index on the log for each active transaction)



316

# DB SERVER ARCHITECTURE: SECONDARY STORAGE AND CACHE

runtime server system: accessed by user queries/updates

- parser: translates into algebra, determines the required relations + indexes
- file manager: determines the file/page where the requested data is stored
- buffer/cache manager: provides relevant data in the cache
- query/update processing: uses only the cache

Cache (main memory): pagewise organized

- Accessed pages are fetched into the cache
- pages are also changed in the cache
- and written to the database later ...

Secondary Storage (Harddisk): pagewise organized

- data pages with tuples
- index pages with tree indexes (see later)
- database log etc. (see later)

### CACHE VS. MATERIALIZATION IN SECONDARY MEMORY

- operations read and write to cache
- contents of the cache is written ("materialized") in secondary storage at "unknown" timepoints
- if a page is moved out from the cache, its modifications are materialized
- write immediate: updates are immediately written to the DB: "simple" power failure cannot lead to redo situations; aborted transactions and power failures require to undo materialized updates in the DB.
- write to DB (at latest) at commit time.
   then, "simple" power failure can still not lead to redo situations
- undo-avoiding:

write ("materialize") updates to the database only (at or) after committing.

- then, aborted transactions are only concerned with the cache (recall that strict 2PL is assumed which prohibits *dirty reads*)
- any power failure ore media crash cannot lead to undo situations (only committed data in DB)

$T_1$ : BC	OT LA RA	WA CO UA			
$T_2$ :	BOT LB RB		LA RA <mark>WB</mark>	CO UA UB	
<b>T</b> <sub>3</sub> :			BOT L	CRC WC	
Buffers:					
<b>T</b> 1:		$A:f_1(a_0)$			
$T_2$ :			$B:f_2(f_1(a_0),b_0)$		
<b>T</b> 3:				$C:f_3(c_0)$	
Databas	se:				
A:	$A_0$	$f_1(A_0)$			
B:	$B_0$		$f_2(f_1(a_0),b_0)$		
C:	C <sub>0</sub>			$f_3(c_0)$	
og:					

### **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.

(Recall that due to strict 2PL, no other transaction could read values that have been written by T)

320

### System Errors

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

- $redone := \emptyset$  and  $undone := \emptyset$ .
- process the logfile backwards until end, or *redone* ∪ *undone* contains all database objects.
   For every entry (*T*, *X*, *X*<sub>old</sub>, *X*<sub>new</sub>):
  - 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.

Exam Consid	<b>ple 6.19</b> der again Example	6.18.				
(write	e-immediate, no un	do-avoiding)		Sys.ei	rror	state after restart
$T_1$ :	BOT LA RA	WA CO UA				
$T_2$ :	BOT LB RB		LA RA <mark>WB</mark>	CO UA UB		
<b>T</b> <sub>3</sub> :			BOTLC	RC	WC	
Buffe	ers:					
$T_1$ :		$A:f_1(a_0)$				
$T_2$ :			$B:f_2(f_1(a_0),b_0)$			
<i>T</i> <sub>3</sub> :				$C : f_3($	(c <sub>0</sub> )	
Data	base:					
<i>A:</i>	$A_0$	$f_1(A_0)$				$f_1(A_0)$
B:	$B_0$		$f_2(f_1(a_0),b_0)$			$f_2(f_1(a_0), b_0)$
C:	C <sub>0</sub>			$f_3($	$(c_0)$	$c_0$
Log: $(T_1, be)$ $(T_3, be)$	$gin), (T_2, begin), (T$ $gin), (T_2, CO), (T_3,$	$egin{aligned} & f_1, A, a_0, f_1(a_0) \ & C, c_0, f_3(c_0)) \end{aligned}$	$), (T_1, CO), (T_2, B)$	$b_0, f_2(f_1(a_0)), b_0$	$b_0)),$	

322

### Logging Requirements

### Log granularity:

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

### Example 6.20

Assume locking at the tuple level, and logging at the relation level, and two transactions:

 $T_1:\ldots,insert(p(1)),\ldots,eot$  $T_2:\ldots,insert(p(2)),\ldots,eot$ 

and the Schedule  $BOT(T_1), \ldots, T_1 : Lp(1), T_1 : insert(p(1)), BOT(T_2), T_2 : Lp(2), T_2 : insert(p(2)), commit(T_2), \ldots, abort(T_1)$ 

The resulting log (initial state of p is  $p_0$ ) is

 $(T_1, begin), (T_1, p, p_0, p_0 \cup \{1\}), (T_2, begin), (T_2, p, p_0 \cup \{1\}, p_0 \cup \{1\} \cup \{2\}), (T_2, commit)$ 

Then the undo operation of  $T_1$  will erase the result of  $T_2$  by resetting p to  $p_0$ .

### 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

324

### 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, keys are obtained, and referential integrity constraints).





# 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.

Pizza-Service								
<u>Name</u>	Address	Product	Number	Price				
Meier	Göttingen	Pizza	10	5.00				
Meier	Göttingen	Lasagne	15	6.00				
Meier	Göttingen	Salad	20	3.00				
Müller	Kassel	Pizza	12	5.00				
Müller	Kassel	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)

328

# Example 7.1 (Continued) *Refined Schema:*

Customer					
<u>Name</u>	Address				
Meier	Göttingen				
Müller	Kassel				

Produ	ıct
Product	Price
Pizza	5.00
Lasagne	6.00
Salad	3.00

Shipment'						
<u>Name</u>	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
- criterion when to decompose a relation schema (and when a decomposition is equivalent) (based on (1))
- measure for "quality" of a schema (in terms of (1))

330

# 7.1 Functional Dependencies

• Data dependencies that describe a functional relationship.

Let  $\overline{V}$  a set of attributes and  $r \in \text{Rel}(\overline{V})$ ,  $\overline{X}, \overline{Y} \subseteq \overline{V}$ . r satisfies the **functional dependency (FD)**  $\overline{X} \to \overline{Y}$  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  $r \in \text{Rel}(\bar{V})$ ).

Refined Definition of "Relation Schema"

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

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

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

### 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

• Does a "good" ER-model already guarantee all desirable properties of the relational model?

# NO

(at least not completely - The more exact the ER model, the better the preliminary relational schema)

• is a formal dependency analysis necessary?

# YES

• theory: based on normal forms of relational schemata





### 7.1.2 Functional Dependency Theory

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

### **Definition 7.1**

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

### **Definition 7.2**

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

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

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

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

336

# 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  $\overline{V}$  and  $r \in \operatorname{Sat}(\overline{V}, \mathcal{F})$ .

(A1) Reflexivity: If  $\overline{Y} \subseteq \overline{X} \subseteq \overline{V}$ , then r satisfies  $\overline{X} \to \overline{Y}$ .

(A2) Augmentation: If  $\overline{X} \to \overline{Y} \in \mathcal{F}$  and  $\overline{Z} \subseteq \overline{V}$ , 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.

П

# 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  $\overline{X} \subseteq \overline{V}$ ,  $\overline{X}^+$  is the set of all  $A \in \overline{V}$  such that  $\overline{X} \to A$  can be derived by the Armstrong axioms.  $\overline{X}^+$  is called the (Attribute-)closure of  $\overline{X}$  (wrt.  $\mathcal{F}$ ).

### Exercise 7.1

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

338

### Proof of Theorem 7.1: correctness is obvious.

Completeness: it has to be shown that if  $\bar{X} \to \bar{Y} \in \mathcal{F}^+$ , then  $\bar{X} \to \bar{Y}$  can be derived by (A1)–(A3) from  $\mathcal{F}$ .

It will be shown: if  $\bar{X} \to \bar{Y}$  is not derivable by (A1)–(A3), then  $\bar{X} \to \bar{Y} \notin \mathcal{F}^+$ , i.e., there is an  $r \in Sat(\bar{V}, \mathcal{F})$  that does not satisfy  $\bar{X} \to \bar{Y}$ .

Assume  $\bar{X} \rightarrow \bar{Y}$  cannot be derived. Consider a relation r as below:

1 1 ... 1 1 1 ... 1 1 1 ... 1  $0 0 \dots 0$ attributes in  $\bar{X}^+$  all other attributes

- (i) First it will be shown that r satisfies  $\mathcal{F}$ : Assume that there is a  $\overline{Z} \to \overline{W} \in \mathcal{F}$ , such that r does not satisfy  $\overline{Z} \to \overline{W}$ . This is only possible if  $\overline{Z} \subseteq \overline{X}^+$  and  $W \not\subseteq \overline{X}^+$ . Since  $\overline{Z} \subseteq \overline{X}^+$ , there is  $\overline{X} \to \overline{Z}$  and  $\overline{Z} \to W$ , and thus  $W \subseteq \overline{X}^+$ , a contradiction.
- (ii) Next, it will be shown that r does not satisfy  $\bar{X} \to \bar{Y}$ : For any  $\bar{X} \to \bar{Y}$  that is satisfied by  $r, \bar{Y} \subseteq \bar{X}^+$ . This would mean that  $\bar{X} \to \bar{Y}$  can be derived from (A1)–(A3).

### **MEMBERSHIP PROBLEM**

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

### Variant 1 :

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

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

### Example 7.5

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

### Membership Problem (Cont'd)

### Variant 2 :

Goal-oriented approach for  $\bar{X} \to \bar{Y} \in \mathcal{F}^+$ :

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

- start with  $X \to X$  (A1 Reflexivity)
- (A2) allows  $\bar{X} \to \bar{Y} \in \mathcal{F} \Rightarrow \overline{XX} \to \overline{XY} \in \mathcal{F}^+$  which is equivalent to  $\bar{X} \to \overline{XY} \in \mathcal{F}^+$
- for any  $\overline{Z} \supset \overline{X}$  and  $\overline{X} \to \overline{XY} \in \mathcal{F}^+$ , (A2) allows to conclude  $\overline{Z} \to \overline{ZY}$  (A2\*)
- "collect"  $\overline{X}^+$  in this way: derive  $\overline{X} \to \overline{XY_1}$ , then  $\overline{XY_1} \to \overline{XY_2}$  by (A2\*) and apply (A3 transitivity) to them,
- until  $\bar{X} \to \bar{Z} \in \mathcal{F}^+$  for  $\bar{Y} \subset \bar{Z}$ , then derive  $\bar{X} \to \bar{Y} \in \mathcal{F}^+$  by (A1).

### Example 7.6

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

	$X \to Y \in \mathcal{F}$	and derive
(A1)	$AB \to AB$	
(A2*)	$AB \to E$	$AB \rightarrow ABE$
(A2*)	$BE \rightarrow I$	$ABE \rightarrow ABEI$
(A2*)	$E \to G$	$ABEI \rightarrow ABEIG$
(A2*)	$GI \to H$	$ABEIG \rightarrow ABEIGH$
(A3) tra	ansitivity:	$AB \rightarrow ABEIGH$
final st	ep with (A1):	$AB \to GH$

342

### Membership Problem (Cont'd)

• consider each (A2\*) + (A3) step as one:

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

### Theorem 7.2

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

**Example 7.7** Apply the  $\bar{X}^+$ -algorithm to Example 7.6 (same steps). 

### **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  $\overline{X} \to \overline{Y}$  and  $\overline{X} \to \overline{Z} \in \mathcal{F}$ , then r satisfies  $\overline{X} \to \overline{YZ}$ .

(A5) Pseudo-transitivity: If  $\overline{X} \to \overline{Y}$  and  $\overline{WY} \to \overline{Z} \in \mathcal{F}$ , then r satisfies  $\overline{XW} \to \overline{Z}$ .

(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  $\overline{X} \to \overline{YZ}$  and  $\overline{Z} \to \overline{AW} \in \mathcal{F}$ , then r satisfies  $\overline{X} \to \overline{YZA}$ .

#### 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.

• (A8) covers the combination of (A2<sup>\*</sup>) and (A3) (consider  $\overline{W} = \emptyset$ ).

### Example 7.8

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

	Derivation by (A7)–(A8)	Intermediate result $ar{X}_i$ of the $ar{X}^+$ -algorithm
(A7)	$AB \to AB$	$\bar{X}_0 = \{A, B\}$
(A8)	$[AB \rightarrow E]$	
	$AB \rightarrow ABE$	$\bar{X}_1 = \{A, B, E\}$
(A8)	[BE  ightarrow I]	
	$AB \rightarrow ABEI$	$\bar{X}_2 = \{A, B, E, I\}$
(A8)	$[E \rightarrow G]$	
	$AB \rightarrow ABEIG$	$\bar{X}_3 = \{A, B, E, I, G\}$
(A8)	$[GI \rightarrow H]$	
	$AB \rightarrow ABEIGH$	$\bar{X}_4 = \{A, B, E, I, G, H\}$
final s	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  $O(|\mathcal{F}| |\bar{V}|^2)$ : 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.

346

# ASIDE: UNIQUE KEYS

### Theorem 7.3

Let  $\mathcal{F} = \{\bar{X}_1 \to \bar{Y}_1, \dots, \bar{X}_p \to \bar{Y}_p\}$ . 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: next slide)

Note:

- $\overline{Z}_1 \cup \ldots \cup \overline{Z}_p$  contains those attributes that are fd from any other attribute.
- $\overline{V} \setminus (\overline{Z}_1 \cup \ldots \cup \overline{Z}_p)$  contains those attributes that are not fd from any other attribute.
- $\bar{V} \setminus (\bar{Z}_1 \cup \ldots \cup \bar{Z}_p)$  is subset of all keys of a relation.

### Example 7.9

Consider the relation Country(name,code,population, area) with FDs name $\rightarrow$  code,population,area and code $\rightarrow$  name,population,area. Here, name and code are keys.

 $\bar{V} \setminus (...) = \emptyset$ 

# ASIDE: UNIQUE KEYS (CONT'D)

Proof of Theorem 7.3:

"⇒" 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,  $\bar{K} \subseteq \bigcap_{i=1}^p (\bar{V} \setminus \bar{Z}_i)$ . 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  $\bar{K}$ ).

" $\Leftarrow$ " Assume  $\bar{K} = \bar{V} \setminus (\bar{Z}_1 \cup \ldots \cup \bar{Z}_p)$  a superkey. It will be shown 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.

348

# 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  $\bar{X} \rightarrow \bar{Y} \in \mathcal{F}$ ,  $\bar{Y}$  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  $\overline{X} \to A \in \mathcal{F}$  and  $\overline{Z} \subset \overline{X}$ , then  $\mathcal{F} \setminus {\{\overline{X} \to A\} \cup {\{\overline{Z} \to A\}}}$  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.

(Note:  $\mathcal{F}^{min}$  is not necessarily unique).

### Example 7.10

Consider again Example 7.9: {name $\rightarrow$  {code}, name $\rightarrow$  {population}, name $\rightarrow$  {area}, code $\rightarrow$ {name}} and {code $\rightarrow$  {name}, code $\rightarrow$  {population}, code $\rightarrow$  {area}, name $\rightarrow$ {code}} are minimal.

### MINIMAL SETS OF FDS

•  $\mathcal{F}^{min}$  can be computed by the  $\bar{X}^+$ -algorithm (without computing  $\mathcal{F}^+$ ) in polynomial time.

Consider a schema  $R(\bar{V}, \mathcal{F})$  with  $|\bar{V}| = n$  and  $|\mathcal{F}| = f$ .

- 1. Decompose all  $X \to Y \in \mathcal{F}$  such that each right side consists of a single attribute; get  $\mathcal{F}'$  with  $|\mathcal{F}'| \leq nf$  in  $O(f \cdot n)$  steps.
- 2. Delete all  $\varphi \in \mathcal{F}'$  that follow from the others (iteratively), using the  $X^+$  algorithm. Each application of  $X^+$  requires  $O(f \cdot n)$  steps, thus, altogether  $O(f^2 \cdot n^2)$ .
- 3. Delete in each remaining FD  $\{x_1 \dots, x_n\} \to y$  stepwise as many attributes on the left side as possible. For each step, check, whether y is still in the remaining  $\{x_1 \dots, x_k\}^+$ . The  $X^+$ -algorithm is applied  $|\mathcal{F}'| \cdot n = O(f \cdot n^2)$  times, thus, this step is in  $O(f^2 \cdot n^3)$ .
- 4. The algorithm is in  $O(f^2 \cdot n^3)$ , i.e., polynomial.

#### 350

# 7.2 Decomposition of Relation Schemata

In Example 7.1 (Slide 328), a relation has been *decomposed* for yielding a better behavior.

### **Definition 7.5**

• Let  $\overline{V}$  a set of attributes. Then,  $\rho = \{\overline{X}_1, \dots, \overline{X}_n\}$  s.t.  $\overline{X}_1 \cup \dots \cup \overline{X}_n = \overline{V}$  and for each i,  $\overline{X}_i \subseteq \overline{V}$  is a **decomposition** of  $\overline{V}$ .

### Example 7.11

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

*E.g.*,  $\rho = \{\{Name, Address\}, \{Product, Price\}, \{Name, Product, Number\}\}$ . is a decomposition.

### Lemma 7.3

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

Then,

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

П

351

### **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 7.12

*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,Ho).

### **Definition 7.6**

Consider a relation schema  $R(\bar{V}, \mathcal{F})$  and a decomposition  $\rho = {\bar{X}_1, \dots, \bar{X}_n}$  of R.  $\rho$  is **lossless** if and only if for every relation  $r \in Sat(\bar{V}, \mathcal{F})$ ,

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

352

# **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  $\rho = {\bar{X}_1, \dots, \bar{X}_n}$  of R.  $\pi[Z](\mathcal{F}) = {X \to Y \in \mathcal{F}^+ \mid XY \subseteq Z}$  is the projection of  $\mathcal{F}$  to Z.

 $\rho$  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:

If the projections of  $\mathcal{F}^+$  are asserted, the (joined) database contents satisfies  $\mathcal{F}$ .

We will first discuss losslessness.



# 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: (Aho, Beeri, Ullman, TODS 1979)

**Idea:** take a tuple  $(a_1, \ldots, a_n)$ , decompose it according to  $\rho$ . Create a "test table" that represents the requirements of a tuple  $(a_1, \ldots, a_n)$  in the re-join of the decomposed tables. Add the knowledge from the FDs about the attribute values of this tuple. The goal is to show that this tuple must have been already present in the original table.

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  $T_{(i,j)} = b_{ij}$  ("any value").

(see next slide)

### Chase-Algorithm for FDs (Cont'd)

As long as T changes, do the following:

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 their  $\bar{Y}$ -values the same:

- For each  $\bar{Y}$ -column j:
- if one of the symbols is  $a_j$ , then replace every occurrence of the other symbol globally by  $a_j$ .
- if both symbols are of the form  $b_{ij}$ , then replace arbitrarily one of them globally by the other.

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

(since they are known to hold in T, this constrains the occurrences of other values)

**Result:**  $\rho$  is lossless if and only if  $(a_1, \ldots, a_n) \in T$ .

356

### Example 7.14 (Chase)

 $ar{V} = ABCDE$ ,  $\rho = (AD, AB, BE, CDE, AE)$ ;  $\mathcal{F} = \{A \rightarrow B, B \rightarrow D, DE \rightarrow C, E \rightarrow A\}$ 

	A	B	C	D	E		A	B	C	D	E
from AD:	$a_1$	$b_{12}$	$b_{13}$	$a_4$	$b_{15}$		$a_1$	$a_2$	$b_{13}$	$a_4$	$b_{15}$
from AB:	$a_1$	$a_2$	$b_{23}$	$b_{24}$	$b_{25}$	chase	$a_1$	$a_2$	$b_{23}$	$a_4$	$b_{25}$
from BE:	$b_{31}$	$a_2$	$b_{33}$	$b_{34}$	$a_5$	7	$\underline{a_1}$	$\underline{a_2}$	$a_3$	$\underline{a_4}$	$\underline{a_5}$
from CDE:	$b_{41}$	$b_{42}$	$a_3$	$a_4$	$a_5$		$a_1$	$b_{42}$	$a_3$	$a_4$	$a_5$
from AE:	$a_1$	$b_{52}$	$b_{53}$	$b_{54}$	$a_5$		$a_1$	$b_{52}$	$b_{53}$	$b_{54}$	$a_5$

The process is finished when the following table is reached:

B	C	D	E
$a_2$	$a_3$	$a_4$	$b_{15}$
$a_2$	$a_3$	$a_4$	$b_{25}$
$a_2$	$a_3$	$a_4$	$a_5$
$a_2$	$a_3$	$a_4$	$a_5$
$a_2$	$a_3$	$a_4$	$a_5$
	$B$ $a_2$ $a_2$ $a_2$ $a_2$ $a_2$ $a_2$	$\begin{array}{c c} B & C \\ \hline a_2 & a_3 \\ a_2 & a_3 \\ a_2 & a_3 \\ a_2 & a_3 \\ a_2 & a_3 \end{array}$	$\begin{array}{c cccc} B & C & D \\ \hline a_2 & a_3 & a_4 \\ a_2 & a_3 & a_4 \end{array}$

Note that only for columns that do not occur on the right side of a FD, the bs remain.

### Theorem 7.5

The above algorithm for testing losslessness is correct.

### Proof:

Notation:

- for a decomposition  $\rho = \{\bar{X}_1, \dots, \bar{X}_k\}$  of  $\bar{V}$  and a relation r, the re-join of the decomposed tables is denoted by  $m_{\rho}(r) = \bowtie_{i=1}^k \pi[\bar{X}_i](r)$ .
- $T_0$  and  $T^*$  denote the table before and after execution of the algorithm.

The algorithm terminates since the number of different symbols decreases with every step.

(A) It has to be shown that if  $\rho$  is lossless,  $(a_1, \ldots, a_n) \in T^*$ .

Due to the construction of  $T_0$ , each  $\pi[\bar{X}_i](T_0)$  contains a row that consists only of  $a_i$ s. Thus,  $(a_1, \ldots, a_n) \in m_\rho(T_0)$ .

This property is preserved by the chase steps, thus  $(a_1, \ldots, a_n) \in m_\rho(T^*)$ . The chase process also guarantees that  $T^* \in \text{Sat}(\bar{V}, \mathcal{F})$ . From the assumption that  $\rho$  is lossless,  $T^* = m_\rho(T^*)$  and  $(a_1, \ldots, a_n) \in T^*$ .

358

### (B) (uses Relational Calculus)

It will be shown that if  $(a_1, \ldots, a_n) \in T^*$ ,  $\rho$  is lossless.

Consider relations r over  $R(\bar{V})$  (as structures). Consider the formula of the calculus

 $F_0 = (\exists b_{11}) \dots (\exists b_{kn}) (R(w_1) \land \dots \land R(w_k))$ 

where  $w_i$  is the *i*-th row of  $T_0$  and all  $a_i$  and  $b_{jk}$ 's are interpreted as variables. The free variables in  $F_0$  are  $a_1, \ldots, a_n$ . Note that every member  $R(w_i)$  of the conjunction in  $F_0$  corresponds to a projection to  $\bar{X}_i$ . Then,

 $m_{\rho}(r) = \operatorname{answers}(F_0(a_1, \dots a_n))$ .

Consider only relations  $r \in \text{Sat}(\bar{V}, \mathcal{F})$ . Since r satisfies  $\mathcal{F}$ ,

 $F_0(a_1,\ldots a_n) \equiv_{\mathcal{F}} F_1(a_1,\ldots a_n) \equiv_{\mathcal{F}} \ldots \equiv F^*(a_1,\ldots a_n)$ 

where each  $F_i$  corresponds to the table after *i* chase steps. For given *r*, the answer set to  $F^*$  is the same as the answer set to  $F_0$ .

Since  $F^*(a_1, \ldots, a_n)$  is of the form  $(\exists b_{11}) \ldots (\exists b_{km})(R(a_1, \ldots, a_n) \land \ldots)$ , its answer set is a subset (or equal) of r.

Altogether,  $m_{\rho}(r) \subseteq r$ . Since  $m_{\rho}(r) \supseteq r$  by Lemma 7.3,  $m_{\rho}(r) = r$ , i.e.,  $\rho$  is lossless.
# Corollary 7.1 (Decomposition into two relations)

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  $\bar{V}$ .  $\rho$  is lossless if and only if

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

#### **Proof:**

The table T for  $\rho$  is

	$\bar{X}_1 \cap \bar{X}_2$	$\bar{X}_1 \setminus \bar{X}_2$	$\bar{X}_2 \setminus \bar{X}_1$
$\bar{X}_1$	$a \dots a$	$a \dots a$	$b \dots b$
$\bar{X}_2$	$a \dots a$	$b \dots b$	$a \dots a$

- 1. Assume  $(a_1, \ldots, a_n) \in T^*$ . Consider an attribute A whose column contains a b. If the algorithm exchanges it by an a, then  $A \in (\bar{X}_1 \cap \bar{X}_2)^+$ . Due to the assumption that  $(a_1, \ldots, a_n) \in T^*$ , there is one line where this happens for all attributes thus all these attributes are in  $(\bar{X}_1 \cap \bar{X}_2)^+$ .
- 2. Assume (w.l.o.g.) that  $(\bar{X}_1 \cap \bar{X}_2) \to (\bar{X}_1 \setminus \bar{X}_2) \in \mathcal{F}^+$ , i.e.,  $\bar{X}_1 \setminus \bar{X}_2 \subseteq (\bar{X}_1 \cap \bar{X}_2)^+$ . Consider the steps for deriving this by the  $\bar{X}^+$ -algorithm. For each such step there is a corresponding chase-step. Thus, the chase replaces each *b* of an attribute in  $\bar{X}_1 \setminus \bar{X}_2$  by an *a*, leading to  $(a_1, \ldots, a_n) \in T^*$ .

# Example 7.15

Consider again Examples 7.4, 7.12 and 7.13 whith the schema  $attends((Student, Course, Lecturer), \{Course \rightarrow Lecturer\})$ 

- $\rho_1 = \{ \{ Course, Lecturer \}, \{ Student, Course \} \}$  is lossless.
- $\rho_2 = \{\{\text{Student}, \text{Lecturer}\}, \{\text{Student}, \text{Course}\}\}$  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, the functional dependency  $Course \rightarrow Lecturer$  which made the decomposition possible.

# 7.2.2 Dependency Preservation

# Example 7.16

Consider again Examples 7.1 and 7.11 with the schema

Pizza-Service( {Name, Address, Product, Number, Price},

 $\{Name \rightarrow Address, Product \rightarrow Price, (Name, Product) \rightarrow Number\})$ 

and the decomposition

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

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

 $\pi[\textit{Name}, \textit{Address}](\mathcal{F}) \supseteq \{\textit{Name} \rightarrow \textit{Address}\}$ 

 $\pi[\text{Product}, \text{Price}](\mathcal{F}) \supseteq \{\text{Product} \rightarrow \text{Price}\}$ 

 $\pi$ [*Name, Product, Number*]( $\mathcal{F}$ )  $\supseteq$  {(*Name, Product*)  $\rightarrow$  *Number* 

So, all FD's immediately survive.

# Another, abstract Example

# Example 7.17

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

 $\rho$  is dependency-preserving (check whether  $C \rightarrow A$  is preserved).

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

( $\mathcal{F}^+$  contains  $A \to ABC$ ,  $B \to ABC$ ,  $C \to ABC$ )

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

# **DEPENDENCY PRESERVATION**

There are lossless decompositions that are not dependency-preserving:

# Example 7.18

Consider  $R = (\overline{V}, \mathcal{F})$ , where  $\overline{V} = \{$ City, Address, Zip $\}$ , and  $\mathcal{F} = \{($ City, Address $) \rightarrow$ Zip,Zip $\rightarrow$ City $\}$ .

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

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

R	City	Address	Zip		$R_1$	Address	Zip	$R_2$	City	Zip
	FR	Herdern	79106			Herdern	79106		FR	79106
	FR	Flughafen	79110			Flughafen	79110		FR	79110
	FR	Mooswald	79110			Mooswald	79110		S	70629
	S	Flughafen	70629			Flughafen	70629	I		•
lnse The	S  Flughaten $ 70629 $ $ Flughaten  70629 nsert (FR,Herdern,79100) and check the FDs:The original FD (City,Address) \rightarrow Zip is not satisfied.$									

364

... and now to a systematic characterization:
some properties have been identified that should hold for a decomposition,
algorithms have been giving for testing them;
is it possible to express properties of such decompositions based on schema information?

how to find such decompositions?

#### 7.3 **Normal Forms based on FDs**

#### Task:

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

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

366

Non-normalized Da	ıta				
lested Relations:					
	Nested	Languages	;		
	Code	Name	Langua	ges	
	D	Germany	German	100	
	СН	Switzerland	German	65	
			French	18	
			Italian	12	
	÷	:	:		
n-atomic values:	<b>I</b>				
	Produ	ucts			
	Code	GDP	Products		
	D	1452200	steel, coal,	chem	icals, machinery, vehicles
	СН	158500	machinery,	chem	icals, 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 redundancy arises (expressed by functional dependencies).

# Example 7.19

Languages							
Code	Name	Language	Percent				
D	Germany	German	100				
СН	Switzerland	German	65				
СН	Switzerland	French	18				
СН	Switzerland	Italian	12				
:							

$$\mathcal{F} = \{ Code \rightarrow Name,$$
  $\square$   
 $Name \rightarrow Code,$   
 $(Code, Language) \rightarrow Percent,$   
 $(Name, Language) \rightarrow Percent \}$ 

368

# Example 7.20

Economy						
Code	GDP	Product				
D	1452200	steel				
D	1452200	coal				
D	1452200	chemicals				
D	1452200	machinery				
D	1452200	vehicles				
СН	158500	machinery				
СН	158500	chemicals				
СН	158500	watches				
-						

 $\mathcal{F} = \{ \textit{(Code, Product)} \rightarrow \textit{(Code, Product, GDP)}, \textit{ Code} \rightarrow \textit{GDP} \}$ 

*Key: (Code, Product)* 

# 2ND NORMAL FORM (2NF)

- In Example 7.20, the GDP information (e.g., (D, 1452200)) is stored redundantly.
- Problem: Code  $\rightarrow$  GDP, but Code alone is not a key.

2NF 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 the above redundancy.

# **Definition 7.9**

A relation schema  $R = (\overline{V}, \mathcal{F})$  is in 2NF if and only if every non-key attribute A is fully dependent on each candidate key:

• Let  $\overline{K}$  be a candidate key of R, A an attribute that is not contained in any candidate key. Then, there is no  $\overline{X} \subsetneq \overline{K}$  s.t.  $\overline{X} \to A \in \mathcal{F}$ .

# Example 7.21

*Consider again Example 7.20: Split the* Economy *relation into relations* Economy'(<u>Code</u>, GDP) *and* Products(<u>Code</u>, <u>Product</u>).

370

# 2ND NORMAL FORM (CONT'D)

The above example was motivated by normalizing a multivalued attribute.

The same situation can occur when mapping a multivalued relationship inaccurately:

• non-key attributes of one of the participating entity types is mixed with the relationship.

	Name I Student	<b>)</b> - < 0, *	> - attends - < 4, * > - Course
	attends		
<u>Student</u>	Course	room	(Student, Course) is (the only) candidate key.
Alice	Databases	E105	$\mathcal{F} = \{ \text{ Course} \rightarrow \text{Room}, \\ (Student, Course) \rightarrow \text{Room} \}$
Bob	Databases	E105	<ul> <li>The table contains redundancies</li> </ul>
Alice	Telematics	E105	<ul> <li>2NF Decomposition: Separate the</li> </ul>
Carol	Telematics	E105	relationship from the entity
Bob	Programming	E203	Totationip nom the ontity.

# 2ND NORMAL FORM (CONT'D)

Separate the relationship from the entity:

	attends		
<u>Student</u>	<u>Course</u>	room	
Alice	Databases	E105	
Bob	Databases	E105	split
Alice	Telematics	E105	
Carol	Telematics	E105	
Bob	Programming	E203	

a	ttends'	
Student	<u>Course</u>	
Alice	Databases	Na
Bob	Databases	Dat
Alice	Telematics	Tele
Carol	Telematics	Pro
Bob	Programming	

Course						
<u>Name</u>	room					
Databases	E105					
Telematics	E105					
Programming	E203					

Is that all?

No. The idea is clear, but the formulation is not yet perfectly accurate.



- the table is in 2NF
- Lecturer  $\rightarrow$  phone does not violate 2NF because Lecturer is not contained in any candidate key this case is not covered by 2NF.

# **3RD NORMAL FORM (3NF)**

# **Definition 7.10**

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

• For each  $\bar{X} \to A \in \mathcal{F}$  such that A is not contained in any candidate key,  $\bar{X}$  contains a candidate key.

Now, all FDs for non key A must be "complete key  $\rightarrow A$ "

3NF Decomposition: Split again.

Separate the relationship from the entity:

rea	ad_by			read_b	y'		
<u>Course</u>	Lecturer	phone		<u>Course</u>	Lecturer	Lectu	urer
Telematics	Ho	14401	solit <sup>.</sup>	Mobile Comm	Ho	Lecturer	phone
Mobile Comm	Но	14401	opiit.	Telematics	Но	Но	14401
Databases	WM	14412		Databases	WM	WM	14412
SSD&XML	WM	14412		SSD&XML	WM		

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



# BOYCE-CODD NORMAL FORM (BCNF)

In Example 7.19 (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 7.11
A relation schema R = (V̄, F) is in BCNF if and only if for each attribute A:

For each X̄ → A ∈ F such that A ∉ X̄, X̄ contains a key.

Example 7.22
Consider again Example 7.19: 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.

376

# BCNF (CONT'D)

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

# Example 7.23

Consider again Example 7.18:

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

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.

It has been 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  $\overline{K}$  its only key. Assume a FD  $\overline{X} \rightarrow A \in \mathcal{F}$ .

We show that  $\overline{X} \to A$  is trivial (i.e.,  $A \in \overline{X}$ ). 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.
- It will be shown that if  $\bar{X} \to A \in \mathcal{F}^+$  and  $A \notin \bar{X}$ , then  $\bar{X} \to \bar{V} \in \mathcal{F}^+$ .

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

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

# Corollary 7.2

A relation schema  $R = (\overline{V}, \mathcal{F})$  is in BCNF if and only if  $R' = (\overline{V}, \mathcal{F}^+)$  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 X
  <sup>+</sup>-algorithm for each FD X
  <sup>-</sup> → Y
  <sup>-</sup> to check if X
  <sup>-</sup> 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  $\overline{V}$ , and  $\overline{X} \subseteq \overline{V}$ .

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).

380

# 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):  $(R \setminus \overline{Y}) \cap \overline{XY} = \overline{X}, \ \overline{XY} \setminus (R \setminus \overline{Y}) = \overline{Y}$ , and thus  $\overline{X} \to \overline{Y}$ .

... 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  $\bar{X} \rightarrow A \in \mathcal{F}$  that violates the BCNF condition.

- Decomposition of  $\overline{V}$ :  $\rho = (\overline{XA}, \overline{V} A)$  (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 7.24

Let  $\overline{V} = \{C, S, J, D, P\}, \ \mathcal{F} = \{SD \rightarrow P, J \rightarrow S\}.$ 



382

# 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
- For each such FD  $\overline{X} \to A$  that is not preserved, extend  $\rho$  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  $\overline{X} \to A \in \mathcal{F}$  and  $\mathcal{F}$  minimal, there is no  $\overline{Y} \to A$  for any  $\overline{Y} \subset \overline{X}$ . Thus,  $\overline{X}$  is a key for  $\overline{XA}$  and all other FDs over  $\overline{XA}$  are defined only over  $\overline{X}$ . Thus, they cannot violate the 3NF-condition (but the BCNF-condition).

# Example 7.25

Consider again Example 7.24.

 $\bar{V} = \{C, S, J, D, P\}, \mathcal{F} = \{SD \rightarrow P, J \rightarrow S\}$ 

- 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  $\{\bar{X} \to A_1, \ \bar{X} \to A_2, \ \ldots\}$  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  $\overline{K}$  of R and add a schema with format K.
- The 3NF-Synthesis-Algorithm is polynomial in time.
- the resulting  $\rho$  is not necessarily minimal: Consider  $\overline{V} = \{AB\}$  with  $\mathcal{F}^{min} = \{A \to B, B \to A\}$ . 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

- Using  $\mathcal{F}^{min}$ , the generated schemata are in 3NF.
- $\rho$  is dependency-preserving since for every  $\bar{X} \to \bar{Y} \in \mathcal{F}^{min}$ , a format is generated that contains  $\overline{XY}$ .
- $\rho$  is lossless since  $\rho$  contains a key of the original schema. Using this tuple, in  $T^*$  (cf. Theorem 7.5) contains a row that consists of  $a_i$ s:

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.
- $i 1 \rightarrow i$ :  $A_i$  is added to  $\bar{X}^+$  due to a FD  $\bar{Y} \rightarrow A_i$  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  $\bar{X}$ .

386

# 7.4 Join Dependencies and Multivalued Dependencies

# Example 7.26

Consider the following Non-1NF table:

ссо							
Country	Continents	Organizations					
D	Europe	NATO, EU, UN					
TR	Europe, Asia	NATO, UN					
R	Europe, Asia	UN					
USA	N.America	UN					

... expand the groups as before to 1NF ...

# Join Dependencies and Multivalued Dependencies (Cont'd)

# Example 7.26 (Continued)

the expanded table:

	cco							
Country	<u>Continent</u>	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	N.America	UN						

There is some redundancy ... called multivalued dependencies cco satisfies

- country ---> continent and
- country ----> organization.

388

# Join Dependencies and Multivalued Dependencies (Cont'd)

# Example 7.26 (Continued)

ссо				
Country	<u>Continent</u>	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	N.America	UN		

#### Actually, cco is a join of

encompasses			isMerr	nber
Country	<u>Cont.</u>		Country	Org.
D	Europe		D	EU
TR	Europe		D	NATO
TR	Asia	and	D	UN
R	Europe		TR	UN
R	Asia		TR	NATO
USA	N.America		R	UN
			USA	UN
$cco = \pi[$	Country,Cont	( <i>cco</i> ) ⋈	$\pi[Country, 0]$	Org](cco)

= encompasses ⋈ isMember

# JOIN DEPENDENCIES (CONT'D)

Consider a set  $\bar{V}$  of attributes, a relation  $r \in \text{Rel}(\bar{V})$ , and a decomposition  $\rho = \{\bar{X}_1, \dots, \bar{X}_n\}$  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

 $\bar{X}_1 \cap \bar{X}_2 \twoheadrightarrow \bar{X}_1 \setminus \bar{X}_2$ , 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  $\bar{X}_2 = (\bar{X}_1 \cap \bar{X}_2) \cup (\bar{X}_2 \setminus \bar{X}_1)$ .

390

# 7.4.1 4. Normal Form (4NF)

**Goal:** mutually independent facts should not be represented in a single relation.

Consider a relation schema  $R = (\overline{V}, D)$  where D is a set of MVDs and FDs. Let  $D^+$  the closure of 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 7.12**

A relation schema  $R = (\bar{V}, D)$  is in 4NF if and only if for every non-trivial  $\bar{X} \twoheadrightarrow Y \in D^+$ ,  $\bar{X}$  contains a key.

# Example 7.27

Consider again Example 7.26. 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 ⇐ (2NF) ⇐ 3NF ⇐ BCNF ⇐ 4NF (⇐ 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  $r_1 \in \text{Rel}(\bar{X}_1)$  and  $r_2 \in \text{Rel}(\bar{X}_2)$  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

 $\pi[\bar{Y}](r_1) \subseteq \pi[\bar{Y}](r_2) .$ 

394

# 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.
- 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
- 3. Transformation to a relational schema
- 4. Normalization to 3NF

the domain specialists.

- 5. Manual decomposition to 4NF
- 6. enhanced ER design.

# IMPORTANCE OF A CORRECT ER-DESIGN

# Example 7.28

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.
- the universal relation of a database (a broad join of all its relations along all FK-PK references) allows data analysis tools to detect such inter-relational FDs.

396

⇒ Re-Design

# Chapter 8 Relational Database Languages: Relational Calculus

# Overview

- the relational calculus is a specialization of first-order logic, 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.

# Relational Calculus vs FOL

- FOL allows for reasoning, based on a model theory,
- · the relational calculus does not require model theory,
- it is only concerned with *validity* of a formula in a given, fixed model (the database state).

# 8.1 Bridge Section: Motivation and Preparation for the "Deductive Databases" Lecture

- The lecture "Database Theory" or "Deductive Databases" (MSc or advanced BSc) builds upon the "Introduction to Databases" lecture and requires knowledge about First-Order Logic (e.g., courses "Formal Systems" or "Artificial Intelligence")
- for a diagram with the database concepts, notions and buzzwords related to the DBIS lectures, see https://www.dbis.informatik.uni-goettingen.de/Teaching/dbnotions.pdf
- This section summarizes that knowledge and motivates the main idea of the lecture.
- a database can be seen as a purely relational FOL structure
  - predicate symbols of different arities,
  - only 0-ary functions = constants
    - \* in relational DB: these are the literals (numbers, strings, dates ...)
    - \* in object-relational DB: also object identifiers
    - $\star\,$  in RDF: also URIs, which basically serve as object identifiers







400

# **Declarative Querying & Algebraic Semantics**

Query: all pairs *country* (name) and *organization* (name) such that the country is a member of the organization.



- declarative query in SQL and as algebra tree (bottom-up inductive semantics)
- actual naive evaluation would be inefficient.

**Declarative Querying & Algebraic Semantics** 

- algebraically equivalent rewriting of the tree,
- efficient evaluation using internal algorithms (more efficient, but correct wrt. the set-oriented algebraic semantics of the operators) and indexes (physical layer):
- start with ismember, search ismember.country→country.code primary key index, then join results.organization→organization.abbreviation primary key index



# RELATIONAL CALCULUS: LOGIC-BASED DECLARATIVE QUERYING

• positional matching of predicate patterns:

 $\begin{array}{ll} q(\textit{pop}) \equiv & \exists \ cc, cap, capprov, area, \texttt{country}(\texttt{`Germany''}, cc, cap, capprov, \textit{pop}, area). \\ q(\textit{cn}, \textit{on}) \equiv & \exists \ cc, cap, capprov, cpop, ca, abbrev, hq, hqc, hqprov, est, type : \\ & \texttt{country}(\textit{cn}, \textit{cc}, cap, capprov, cpop, ca) \land \\ & \texttt{organization}(abbrev, \textit{on}, hq, hqc, hqprov, est) \land \\ & \texttt{ismember}(\textit{cc}, abbrev, type) \end{array}$ 

- purely declarative
- "conjunctive query", translatable to relational algebra SPJR-query (selection-projection-renaming-join)
- free variables (here, *cn*, *on*) create the result tuples,

```
answer = \{ \{cn/"Germany", on/"Europ.Union"\}, \{cn/"Germany", on/"North.Atl.Tr.Org"\}, \dots, \\ \{cn/"France", on/"Europ.Union"\}, \{cn/"France", on/"North.Atl.Tr.Org"\}, \dots, \\ \vdots \\ \}
```



# Closed-World-Assumption: Negation - not exists

- In databases, all tuples that are not there are implicit negative knowledge
- query from previous slide:
   "all countries such that there is no tuple in the the database that states that the country would be located in Europe"
- $\Rightarrow$  "Negation by default"
- $\Rightarrow$  consistent with the assumption that the database contains complete knowledge.
  - as a first-order/predicate logic interpretation, for all answer bindings β (that bind the variable *cn*),

 $(\mathcal{S},\beta) \models \exists cc, cap, capprov, cpop, ca:$ 

 $country(cn, cc, cap, capprov, cpop, ca) \land \neg \exists perc : encompasses(cc, "Europe", p)$ 

• let  $\varphi$  the conjunction of all facts (=atoms) that are true in the database,

 $\varphi \not\models \exists cc, cap, capprov, cpop, ca:$ 

 $\mathsf{country}(\mathit{cn}, \mathit{cc}, \mathit{cap}, \mathit{capprov}, \mathit{cpop}, \mathit{ca}) \land \neg \exists \mathit{perc} : \mathsf{encompasses}(\mathit{cc}, \mathsf{``Europe''}, p)$ 

since  $\neg \exists perc$  : encompasses(*cc*, "Europe", *p*) *cannot logically be concluded* ("Open World")

# Negation: Safety of Variables

Consider just a binary *isMember* relationship for mondial without the membership type:

 $q(c) \equiv \neg \mathsf{ismember}(c, \mathsf{``EU"})$ 

- what are the answers?
- "USA", "AUS", ..., but also "Moscow", "Berlin", 356000, 3.1415 etc., infinitely many, for which the tuple is not true.
- $\Rightarrow$  depends on the considered *domain*.
- ⇒ every query must be *safe*, i.e., the variables must have a positive occurrence that restricts the possible values:

 $q'(c) \equiv \exists name, c, cap, capprov, cpop, ca :$ country(name, c, cap, capprov, cpop, ca)  $\land \neg ismember(c, "EU")$ 

	$head \leftarrow body$
• SOL	body = FROM WHERE
OQL.	$bead = SELECT \qquad DELETE$
	similar: MODIFY <relname> WHERE, INSERT INTO (SFW)</relname>
• SQL v	views: derive new tuple(s) when body is satisfied
• An SC	QL view must not be recursive (i.e., contain itself in the "body" part)
Datalog:	Queries and Logical Rules
?- count	try(N, _C, _Cap, _CapProv, _Pop, _Area), not isMember(_C, 'EU', _).
wo rules	that together compute for each river, to which sea its water finally flows:
:- inclu	de(mondial).
tc(N,S)	:- river(N,R,L,S,_,_,_,_,_,_,_), not (S = null).
tc(N,S)	:- river(N,R,L,S2,_,_,_,_,_,_,_), not (R = null), tc(R,S).
Filename	: Datalog/tcRivers.P]

The Universal Quantifier in Query Languages

- SQL: EXISTS/NOT EXISTS has been integrated into the SQL syntax (implemented via Join, Minus, Anti-Join)
- The universal quantifier must be rewritten as NOT EXISTS ... WHERE NOT EXISTS ...
- the relational calculus obviously allows it:

 $q(cn) \equiv \exists cc, cap, capprov, pop, area:$ 

 $(country(cn, cc, cap, capprov, pop, area) \land$ 

```
\forall n, prov, cpoplat, long, el : (city(n, cc, prov, cpop, lat, long, el) \rightarrow cpop > 1000000)
```

- Datalog: universal quantifier must be encoded into rules
- XQuery (query language for XML data) has it:

//country[.//city/population

 $\operatorname{and}$ 

```
(every $cp in .//city/population satisfies $cp > 1000000)]/name
```

• note: null values and missing values (in XML) have been ignored here.

408

# **TYPES OF KNOWLEDGE**

- (positive) atomic facts:
  - DB: tuples in an *n*-column table of the database
  - FOL: S = (I, D): for an *n*-ary predicate,  $I(p) \subseteq D^n$
  - atoms in a formula
  - $\Rightarrow$  conjunctions/sets of atomic facts
- negative atomic facts/knowledge:
  - rather "implicit": the *n*-tuples "not there" in a DB or not in I(p).
  - $\Rightarrow$  queries under CWA and  $\mathcal{S} \models \varphi$ .
- · atomic positive conclusions: INSERT into DB, Views
- atomic negative conclusions: DELETE, or inconsistencies

**Disjunctive Knowledge** 

- "p(x) or q(y) does hold"
- cannot be represented by a database or a single FOL interpretation, only by formulas
- $\Rightarrow$  conclusions in "knowledge base"



# Existential Knowledge

- "every country has some city that is its capital (and which is located in this country)"  $\forall x: country(x) \rightarrow \exists y: (city(y) \land hasCapital(x, y) \land located\_in(y, x))$ 
  - SQL: country.capital not null and a foreign-key-to-primary-key reference: country.(code, capital, capprov) references city.(country, name, province) only as a passive constraint, cannot conclude and insert the city (name is not known)
  - ER-Diagram: minCardinality for *capital*, but not that *isCapital*  $\subseteq$  *locatedIn*
  - OWL/Description Logic: *Country*  $\sqsubseteq \exists$  *hasCapital.City* and *isCapitalOf*  $\sqsubseteq$  *locatedIn*
- "everything which is a parent has *some* child (which is a person)"
   ER Diagram: *Parent* is a subclass of *Person*, minCardinality of *hasChild* is 1
   OWL/Description Logic: *Parent* ≡ ∃*hasChild*.*Person*
  - ⇐: SQL: view, FOL: conclude an atom
  - $\Rightarrow$ : SQL: not possible FOL, e.g. tableau calculus use a skolem function and derive hasChild(alice,  $f_{child}$ (alice)) and Person( $f_{child}$ (alice))
- "every person has two parents which are persons"
  - would create/insert infinitely many new objects  $\rightarrow$  needs a blocking strategy
  - in general, created objects may be equal or not (tableau calculus:  $\rightarrow$  branching)

# 8.2 First-Order Logic

The relational calculus is a specialization of first-order logic.

# 8.2.1 Syntax

- each first-order language contains the following distinguished symbols:
  - "(" and ")", logical symbols  $\neg$ ,  $\land$ ,  $\lor$ ,  $\rightarrow$ , 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**  $\Sigma$ .  $\Sigma$  contains **function symbols** and **predicate symbols**, each of them with a given arity.

412

# Aside/Preview: First-Order Modeling Styles

• the choice between predicate and function symbols and different arities allows multiple ways of modeling (see Slide 435).

# 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**; in a relational database these are only the literal values (numbers and strings).
- thus, the database schema  ${\bf R}$  is the signature.

# Syntax (Cont'd)

#### Terms

The set of **terms** over  $\Sigma$ , Term<sub> $\Sigma$ </sub>, 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.

414

# 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  $\theta$  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  $\exists X : F$  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  $F \lor \neg G$ , the **implication** syntax  $F \leftarrow G$  or  $G \rightarrow F$  can be used, and
- $(F \to G) \land (F \leftarrow G)$  is denoted by the **equivalence**  $F \leftrightarrow G$ .

#### 416

# 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",  $X) \land X > 10.000.000$ : X is free.
- $\exists X : (continent("Asia", X) \land X > 10.000.000): X is bound.$ The formula is closed.
- $\exists X : (continent(X, Y)): X \text{ is bound, } Y \text{ is free.}$
- $\forall Y : (\exists X : (continent(X, Y))): X \text{ and } Y \text{ are bound.}$ 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.
- ∃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").

418

# 8.2.2 Semantics

The semantics of first-order logic is given by first-order structures over the signature:

**First-Order Structure** 

A first-order structure S = (I, D) over a signature  $\Sigma$  consists of a nonempty set D (domain; often also denoted by U (universe)) and an interpretation I of the signature symbols over 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  $I(f) : \mathcal{D}^n \to \mathcal{D}$ (note that for relational databases, there are no function symbols with arity > 0)
- every *n*-ary predicate symbol *p* to an *n*-ary relation  $I(p) \subseteq \mathcal{D}^n$ .

#### General:

- · constants are interpreted by elements of the domain
- predicate symbols and function symbols are *not* mapped to domain objects, but to relations/functions over the domain.
  - $\Rightarrow$  First-order logic cannot express relations/relationships between predicates/functions.

Aside/Preview: First-Order-based Semantic Styles

- There are different frameworks that are based on first-order logic that specialize/simplify FOL (see Slide 435).
- Higher-Order logics allow to make statements about predicates and/or functions by higher-order predicates.

420

# First-Order Structures: An Example

#### Example 8.1 (First-Order Structure)

Signature: constant symbols: zero, one, two, three, four, five predicate symbols: green/1, red/1, sees/2 function symbols: to\_right/1, plus/2 Structure S: *Domain*  $\mathcal{D} = \{0, 1, 2, 3, 4, 5\}$ Interpretation of the signature:  $I(zero) = 0, I(one) = 1, \dots, I(five) = 5$  $I(green) = \{(2), (5)\}, I(red) = \{(0), (1), (3), (4)\}$  $I(sees) = \{(0,3), (1,4), (2,5), (3,0), (4,1), (5,2)\}$  $I(to\_right) = \{ (0) \mapsto (1), (1) \mapsto (2), (2) \mapsto (3), \}$  $(3) \mapsto (4), (4) \mapsto (5), (5) \mapsto (0)$  $I(plus) = \{(n,m) \mapsto (n+m) \mod 6 \mid n,m \in \mathcal{D}\}$ **Terms:** one, to right(four), to right(to right(X)), to right(to right(to right(four))), plus(X, to\_right(zero)), to\_right(plus(to\_right(four), five))) Atomic Formulas: green(one), 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  ${\cal S}$  over  ${\bf R}$
- $\mathcal{D}$  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(encompasses)* (which we also write as S(encompasses)).

#### For Databases:

- no function symbols with arity > 0
- constants are interpreted "by themselves": I(4) = 4, I("Asia") = "Asia"
- · care for domains of attributes.

422

#### Evaluation of Terms and Formulas

Terms and formulas must be **evaluated** under a given interpretation – i.e., wrt. a given database state S.

- Terms can contain variables.
- variables are not interpreted by S.

A variable assignment over a universe  $\mathcal{D}$  is a mapping

 $\beta: Variables \to \mathcal{D}$ .

For a variable assignment  $\beta$ , a variable X, and  $d \in D$ , the **modified** variable assignment  $\beta_X^d$  is identical with  $\beta$  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

- wrt. a given structure  $\mathcal{S} = (I, \mathcal{D})$ , and
- wrt. a given variable assignment  $\beta$ .

Every structure S together with a variable assignment  $\beta$  induces an evaluation S of terms and predicates:

- Terms are mapped to elements of the universe: S : Term<sub> $\Sigma$ </sub> ×  $\beta \rightarrow D$
- (Closed) formulas are true or false in a structure:  $S : Fml_{\Sigma} \times \beta \rightarrow \{true, false\}$

For Databases:

- $\Sigma$  is a purely relational signature,
- $\mathcal{S}$  is a database state for  $\Sigma$ ,
- no function symbols with arity > 0, no nontrivial terms,
- · constants are interpreted "by themselves".

424

# **Evaluation of Terms**

$$\begin{split} \mathcal{S}(x,\beta) &:= \beta(x) \quad \text{for a variable } x \ , \\ \mathcal{S}(c,\beta) &:= I(c) \quad \text{for any constant } c \ . \\ \mathcal{S}(f(t_1,\ldots,t_n),\beta) &:= (I(f))(\mathcal{S}(t_1,\beta),\ldots,\mathcal{S}(t_n,\beta)) \end{split}$$

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:  $\beta = \emptyset$ .
- $\mathcal{S}(one, \emptyset) = I(one) = 1$
- $S(to\_right(four), \emptyset) = I(to\_right(S(four, \emptyset))) = I(to\_right(4)) = 5$
- $S(to\_right(to\_right(four))), \emptyset) = I(to\_right(S(to\_right(to\_right(four)), \emptyset))) = I(to\_right(I(to\_right(S(to\_right(four), \emptyset))))) = I(to\_right(I(to\_right(I(to\_right(S(four)), \emptyset))))) = I(to\_right(I(to\_right(I(to\_right(4), \emptyset))))) = I(to\_right(I(to\_right(5)))) = I(to\_right(0)) = 1$

#### Example 8.3 (Continued)

- Let  $\beta = \{X \mapsto 3\}$ .  $S(to\_right(to\_right(X)), \beta) = I(to\_right(S(to\_right(X), \beta))) =$   $I(to\_right(I(to\_right(S(X, \beta))))) = I(to\_right(I(to\_right(\beta(X))))) =$  $I(to\_right(I(to\_right(3)))) = I(to\_right(4)) = 5$
- Let  $\beta = \{X \mapsto 3\}$ .  $S(plus(X, to\_right(zero)), \emptyset) = I(plus(S(X, \beta), S(to\_right(zero), \beta))) = I(plus(\beta(X), I(to\_right(S(zero, \beta))))) = I(plus(3, I(to\_right(0)))) = I(plus(3, 1)) = 4$

426

# **EVALUATION OF FORMULAS**

Formulas can either hold, or not hold in a database state.

**Truth Value** 

Let *F* a formula, *S* an interpretation, and  $\beta$  a variable assignment of the free variables in *F* (denoted by free(F)).

Then we write  $S \models_{\beta} F$  if "*F* is true in *S* wrt.  $\beta$ ".

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  $a_i, 1 \le i \le k$  are constants,

```
R(a_1,\ldots,a_k) is true in S if and only if (I(a_1),\ldots,I(a_k)) \in S(R).
```

Otherwise,  $R(a_1, \ldots, a_k)$  is **false** in S.

Base Case: Atomic Formulas

The **truth value** of an atom  $R(t_1, \ldots, t_k)$ , where  $t_i, 1 \le i \le k$  are terms, is given as

 $\mathcal{S} \models_{\beta} R(t_1, \ldots, t_k)$  if and only if  $(\mathcal{S}(t_1, \beta), \ldots, \mathcal{S}(t_k, \beta)) \in \mathcal{S}(R)$ .

For Databases:

• the  $t_i$  can only be constants or variables.

428

# **TRUTH VALUES OF FORMULAS: INDUCTIVE DEFINITION**

- $t_1 \theta t_2$  with  $\theta$  a comparison operator in  $\{=, \neq, \leq, <, \geq, >\}$ :  $S \models_{\beta} t_1 \theta t_2$  if and only if  $S(t_1, \beta) \theta S(t_2, \beta)$  holds.
- $\mathcal{S} \models_{\beta} \neg G$  if and only if  $\mathcal{S} \not\models_{\beta} G$ .
- $\mathcal{S} \models_{\beta} G \land H$  if and only if  $\mathcal{S} \models_{\beta} G$  and  $\mathcal{S} \models_{\beta} H$ .
- $\mathcal{S} \models_{\beta} G \lor H$  if and only if  $\mathcal{S} \models_{\beta} G$  or  $\mathcal{S} \models_{\beta} H$ .
- (Derived; cf. next slide)  $\mathcal{S} \models_{\beta} F \to G$  if and only if  $\mathcal{S} \models_{\beta} \neg F$  or  $\mathcal{S} \models_{\beta} G$ .
- $\mathcal{S} \models_{\beta} \forall XG$  if and only if for all  $d \in \mathcal{D}$ ,  $\mathcal{S} \models_{\beta_X^d} G$ .
- $\mathcal{S} \models_{\beta} \exists XG$  if and only if for some  $d \in \mathcal{D}, \mathcal{S} \models_{\beta_{\mathbf{v}}^d} G$ .

#### **Derived Boolean Operators**

There are some minimal sets (e.g.  $\{\neg, \land, \exists\}$ ) of boolean operators from which the others can be derived:

- The implication syntax F → G is a shortcut for ¬F ∨ G (cf. Slide 416):
  S ⊨<sub>β</sub> F → G if and only if S ⊨<sub>β</sub> ¬F or S ⊨<sub>β</sub> G.
  "whenever F holds, also G holds" this is called *material implication* instead of "causal implication".
  Note: *if* F implies G causally in a scenario, then all (possible) states satisfy F → G.
- note that  $\land$  and  $\lor$  can also be expressed by each other, together with  $\neg$ :  $F \land G$  is equivalent to  $\neg(\neg F \lor \neg G)$ , and  $F \lor G$  is equivalent to  $\neg(\neg F \land \neg G)$ .
- The quantifiers  $\exists$  and  $\forall$  are in the same way "dual" to each other:  $\exists x : F$  is equivalent to  $\neg \forall x : (\neg F)$ , and  $\forall x : F$  is equivalent to  $\neg \exists x : (\neg F)$ .
- Proofs: exercise. Show e.g. by the definitions that whenever  $S \models_{\beta} \exists x : F$  then  $S \models_{\beta} \neg \forall x : (\neg F)$ .

430

# Example 8.4 (Evaluation of Atomic Formulas)

Consider again Example 8.1.

- For variable-free formulas, let  $\beta = \emptyset$
- $S \models_{\emptyset} green(one) \Leftrightarrow S(one) \in I(green) \Leftrightarrow (1) \in I(green) which is not the case.$ Thus,  $S \not\models_{\emptyset} green(one)$ .
- $\mathcal{S} \models_{\emptyset} red(to\_right(to\_right(to\_right(three)))) \Leftrightarrow$

 $(\mathcal{S}(to\_right(to\_right(to\_right(three))), \emptyset)) \in I(red) \iff (0) \in I(red)$ 

which is the case. Thus,  $S \models_{\emptyset} red(to\_right(to\_right(to\_right(three))))$ .

- 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,  $\beta = \{X \mapsto 3, Y \mapsto 5\}$ .  $\mathcal{S} \models_{\beta} sees(X, to\_right(Y)) \Leftrightarrow (\mathcal{S}(X, \beta), \mathcal{S}(to\_right(Y), \beta)) \in I(sees) \Leftrightarrow (3, 0) \in I(sees)$ which is the case.
- $S \models_{\beta} plus(to\_right(to\_right(four)), to\_right(one)) = to\_right(three) \Leftrightarrow$  $S(plus(to\_right(to\_right(four)), to\_right(one)), \emptyset) = S(to\_right(three), \emptyset) \Leftrightarrow 2 = 4$ which is not the case.
#### 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  $S \models_{\emptyset_X^d} red(X) \Leftrightarrow$  there is a  $d \in \mathcal{D}$  s.t.  $S \models_{\{X \mapsto d\}} red(X)$ Since we have shown above that  $S \models_{\emptyset} red(6)$ , this is the case.

•  $\mathcal{S} \models_{\emptyset} \forall X : green(X) \Leftrightarrow$ 

for all  $d \in \mathcal{D}$ ,  $\mathcal{S} \models_{\emptyset_{\mathcal{X}}^d} green(X) \Leftrightarrow$  for all  $d \in \mathcal{D}$ ,  $\mathcal{S} \models_{\{X \mapsto d\}} green(X)$ 

Since we have shown above that  $S \not\models_{\emptyset} green(1)$  this is not the case.

- $S \models_{\emptyset} \forall X : (green(X) \lor red(X)) \Leftrightarrow$  for all  $d \in D$ ,  $S \models_{\{X \mapsto d\}} (green(X) \lor red(X))$ . One has now to check whether  $S \models_{\{X \mapsto d\}} (green(X) \lor red(X))$  for all  $d \in domain$ . We do it for d = 3:
  - $\mathcal{S} \models_{\{X \mapsto 3\}} (green(X) \lor red(X)) \Leftrightarrow$

 $\mathcal{S} \models_{\{X \mapsto 3\}} green(X) \text{ or } \mathcal{S} \models_{\{X \mapsto 3\}} red(X) \Leftrightarrow$ 

 $(\mathcal{S}(X, \{X \mapsto 3\})) \in I(green) \text{ or } (\mathcal{S}(X, \{X \mapsto 3\})) \in I(red) \Leftrightarrow$ 

 $(3) \in I(green) \text{ or } (3) \in I(red)$ 

which is the case since  $(3) \in I(red)$ .

• Similarly,  $\mathcal{S} \not\models_{\emptyset} \forall X : (green(X) \land red(X))$ 

432

#### Some Notions

Consider a formula F with some free variables.

- S is a model for F under  $\beta$  if  $S \models_{\beta} F$ .
- (for closed formulas: S is a model for F if  $S \models F$ )
- *F* is *satisfiable* if *F* has some model (e.g.,  $F = \exists x, y : (p(x) \land q(x, y))$  is satisfiable).
- *F* is *unsatifisfiable* if *F* has no model (e.g.,  $F = \exists x : (p(x) \land \neg p(x) \text{ is unsatisfiable})$
- *F* is *valid* (german: "allgemeingültig") if *F* holds in every structure:
  (e.g., *F* = (∀x : (p(x) → q(x)) ∧ ∀y : (q(y) → r(y))) → ∀z : (p(z) → r(z))) is valid)
  Application: verification of a system has the goal to show that φ → ψ is valid where φ is a formula that contains the specification (usually a large conjunction) and φ is a conjunction of guaranteed properties.
- two FOL formulas F and G are *equivalent*, F = G if every model of F is also a model of G and vice versa.
- a FOL formula *F* entails a FOL formula *G*, *F* ⊨ *G* if every model of *F* is also a model of *G*.
   (note the overloading of ⊨ for *S* ⊨ *F* and *F* ⊨ *G*).

#### Example 8.6

For the following pairs F and G of formulas, check whether one implies the other (if not, give a counterexample), and whether they are equivalent:

- 1.  $F = (\forall x : p(x)) \lor (\forall x : q(x)), \quad G = \forall v : (p(v) \lor q(v)).$
- **2.**  $F = \forall x : ((\exists y : p(y)) \rightarrow q(x)), \quad G = \forall v, \forall w : p(v) \rightarrow q(w).$
- **3.**  $F = \forall x : \exists y : p(x, y), \quad G = \exists v : \forall w : p(v, w).$

434

# 8.3 FOL-based Modeling Styles and Frameworks

- Full FOL allows for several restrictions, shortcuts and extensions
- variants developed depending on the application and the intended reasoning mechanisms.

#### Recall

- note: the FOL signature is disjoint from the domain D, e.g. germany is a constant symbol, mapped to the element germany ∈ D.
- · each FOL signature consists of
  - predicate symbols
    - \* 0-ary predicates: "boolean predicates", just being interpreted as true/false (formally  $I(p_0) \subseteq \mathcal{D}^0$ , where  $\mathcal{D}^0 = 1$  means true, while  $\emptyset$  means false).
    - \* *n*-ary predicates, interpreted as  $\mathcal{I}(p) \subseteq \mathcal{D}^n$ .
  - function symbols
    - \* 0-ary functions: constants, interpreted by elements of the domain.
       (formally *I*(*c*) : D<sup>0</sup> → D, e.g. for the constant germany: *I*(germany) : () → germany;
       S(germany) = I(germany()) = germany)
    - \* *n*-ary functions, interpreted as  $\mathcal{I}(f) : \mathcal{D}^n \to \mathcal{D}$ .

П

# 8.3.1 FOL with (atomic) Datatypes

Common extension:  $FOL(D_1, ..., D_n)$  where  $D_1, ..., D_n$  are datatypes like strings, numbers, dates.

- for these, the values are both 0-ary constant symbols and elements of the domain,
- appropriate predicates and functions are contained in the signature and as built-in predicates and functions (i.e., are not explicitly mentioned when giving an interpretation).

#### Example 8.1 revisited

Example 8.1 can be formulated in FOL(*INT*):

- integers  $0, 1, 2, \ldots \in \Sigma$  as constant symbols (instead of *one, two, ...*).
- I(0) = 0, I(1) = 1, ... is implicit.
- no interpretation of the constant symbols one, two, ... required.
- function +/2 (i.e., binary function "+") instead of *plus/2*, its interpretation comes implicitly from integers.
- interpretation of user-defined predicates green, sees, to\_right as before (over the domain  $D \supseteq INT$ ).

436

# 8.3.2 Purely Relational Object-Oriented Modeling

- Closely related with the ER Model:
- the domain  $\mathcal{D}$  contains instances/individuals/"resources" *germany*, *berlin*, ... and datatype literals.
- - Entity types = Classes: unary predicates  $germany \in I(Country), \quad berlin \in I(City), \quad eu \in I(Organization).$ 
  - Attributes: binary predicates
     (germany, "Germany") ∈ I(name),
     (berlin, "3472009") ∈ I(population)
  - Relationships: binary predicates (germany, berlin)  $\in I(capital)$ , (germany, eu)  $\in I(isMember)$ .
- closely related: RDF Resource Description Framework as the data model underlying the Semantic Web (cf. Slide 440).
- closely related: Specific family of logics called "Description Logic" as a *decidable* subset of FOL (cf. Slide 441)

#### Examples

The following sets specify answers to sample queries:

• Names of all countries such that there is a city with more than 1,000,000 inhabitants in the country:

```
\{n \mid \ \exists x: \mathsf{Country}(x) \land \mathsf{name}(x,n) \land \\
```

 $\exists y, p : (\mathsf{City}(y) \land \mathsf{inCountry}(x, y) \land \mathsf{population}(y, p) \land p > 1,000,000) \}$ 

• Names of all countries such that all its cities have more than 1,000,000 inhabitants:

```
\{n \mid \exists x: \mathsf{Country}(x) \land \mathsf{name}(x, n) \land
```

 $\forall y : (\mathsf{City}(y) \land \mathsf{inCountry}(x, y) \to \exists p : (\mathsf{population}(y, p) \land p > 1,000,000)) \}$ 

- Names of all countries such that the capital of the country has more than 1,000,000 inhabitants:
  - $\{n \mid \ \exists x: \mathsf{Country}(x) \land \mathsf{name}(x,n) \land \\$

 $\exists y, p : (\mathsf{City}(y) \land \mathsf{capital}(x, y) \land \mathsf{population}(y, p) \land p > 1,000,000) \}$ 

 Names of all countries such that the country is a member of the organization with abbreviation "EU":

```
\{n \mid \exists x: \mathsf{Country}(x) \land \mathsf{name}(x, n) \land
```

 $\exists o : (\mathsf{Organization}(o) \land \mathsf{abbrev}(o, \mathsf{``EU''}) \land \mathsf{isMember}(x, o)) \}$ 

438

#### Problem

⇒ attributed relationships (like isMember with membertype) can only be modeled via reification.

#### Example

 $(delnEU) \in I(Membership),$ 

 $(delnEU, germany) \in I(ofCountry).$  $(delnEU, eu) \in I(inOrganization).$ 

(*delnEU*, "full member")  $\in I(\text{memberType})$ .

Names of all countries such that the country is a member of the organization with abbreviation "EU":

```
\{n \mid \exists x : (\mathsf{Country}(x) \land \mathsf{name}(x, n) \land
```

 $\exists o, m, t : (\operatorname{Organization}(o) \land \operatorname{abbrev}(o, ``\mathsf{EU''}) \land \\$ 

 $\land \mathsf{Membership}(m) \land \mathsf{ofCountry}(m, x) \land \mathsf{inOrganization}(m, o) \land \mathsf{memberType}(m, t))) \; ]$ 

# **RDF – RESOURCE DESCRIPTION FRAMEWORK**

- most prominent Semantic Web data model.
- graph-based: objects and literals are nodes, properties are the edges.
- instance data represented by (subject predicate object) triples that can be seen as unary (class membership) and binary (properties and relationships) predicates:
  - :germany a mon:Country. :germany mon:name "Germany" :germany mon:population 83536115.
- Country(germany)
- name(germany, "Germany")
- population(germany, 83536115)
- :germany mon:capital :berlin.
- capital(germany, berlin)
- optional: XML serialization
- domain: URIs and literals (using the XML namespace concept)
  - URIs serve as constant symbols and (web-wide) object/resource identifiers,
  - property and class names are also URIs.

440

# **DESCRIPTION LOGICS**

- traditional framework, became popular as a base for the Semantic Web,
- subset of FOL where the formulas are restricted,
- $\Rightarrow$  modular family of logics, most of which are decidable.
  - special syntax that can be translated into the 2-variable fragment of FOL (decidable).
  - focus of DL is on the definition of concepts:

#### $CoastCity \equiv City \sqcap \exists locatedAt.Sea$ .

FOL:  $\forall x : CoastCity(x) \leftrightarrow City(x) \land \exists y : (IocatedAt(x, y) \land Sea(y)).$ 

# 8.3.3 FOL Object-Oriented Modeling with Functions

- S = (I, D) as follows:
- the domain  $\mathcal{D}$  contains elements germany, berlin, ... and datatype literals
- Predicates Country/1, City/1, Organization/1, ismember/2 etc. as before,
- functions capital/1, headq/1, population/1 for *functional* attributes and relationships: (*germany*) → *berlin* ∈ *I*(capital),
   (*eu*) → *brussels* ∈ *I*(headq),
   (*berlin*) → 3472009 ∈ *I*(population).
- · some example formula that evaluates to true:

```
\mathcal{S} \models \exists o, c : \mathsf{Organization}(o) \land \mathsf{name}(o) = \mathsf{``Europ.Union''} \land \mathsf{isMember}(c, o) \land \mathsf{headq}(o) = \mathsf{capital}(c)
```

(FOL with equality)

442

## 8.3.4 Relational Calculus ("Domain Relational Calculus")

- The signature Σ is a relational database schema R = {R<sub>1</sub>,..., R<sub>n</sub>}.
   ⇒ everything is modeled by predicates.
- the domain consists only of *datatype literals* (strings, numbers, dates, ...).
- constant symbols are the literals themselves, with e.g. I(3) = 3 and I("Berlin") = "Berlin".
- ⇒ a relational database state S = (I, (Strings + Numbers + Dates)) over **R** is an interpretation of **R**. For every relation name  $R_i \in \mathbf{R}$ ,  $I(R_i)$  is a finite set of tuples: ("Germany", "D", 356910, 83536115, "Berlin", "Berlin")  $\in I(\text{country})$ , ("D", "Europe", 100)  $\in I(\text{encompasses})$ .
  - I (and by this, also S) can be described as a *finite* set of ground atoms over predicate symbols (= relation names): country("Germany", "D", 356910, 83536115, "Berlin", "Berlin"), encompasses("D", "Europe", 100).
  - the purely value-based "modeling" without individuals/object identifiers/0-ary constant symbols requires the use of primary/foreign keys.
  - semantics and model theory as in traditional FOL; quantifiers range over the literals – "Domain Relational Calculus"
  - usage: theoretical framework for queries; mapped to nonrecursive Datalog with negation.

#### Examples

The following sets specify answers to sample queries:

 Names of all countries such that there is a city with more than 1,000,000 inhabitants in the country:

 $\{n \mid \exists cc, ca, cp, cap, capprov : \mathsf{Country}(n, cc, ca, cp, cap, capprov) \land \\ \exists ctyn, ctyprov, ctypop, lat, long : \\ (\mathsf{City}(ctyn, ctyprov, cc, ctypop, lat, long) \land ctypop > 1,000,000) \}$ 

- Names of all countries such that all its cities have more than 1,000,000 inhabitants:
  - $\{n \mid \exists cc, ca, cp, cap, capprov : \mathsf{Country}(n, cc, ca, cp, cap, capprov) \land \\ \forall ctyn, ctyprov, ctypop, lat, long : \\ (\mathsf{City}(ctyn, ctyprov, cc, ctypop, lat, long) \rightarrow ctypop > 1,000,000) \}$
- Names of all countries such that the country is a member of the organization with name "Europ.Union":
  - $\{n \mid \exists cc, ca, cp, cap, capprov : \mathsf{Country}(n, cc, ca, cp, cap, capprov) \land \\$

 $\exists abbr, hq, hqp, hqc, est, t:$ 

 $(Organization(abbr, "Europ.Union", hq, hqc, hqp, est) \land isMember(cc, abbr, t)) \}$ 

444

# 8.3.5 Relational Calculus ("Tuple Relational Calculus")

- · Logical connectives and quantifiers as in FOL,
- syntax and semantics different from FOL: quantifiers range over tuples "Tuple Relational Calculus"
- Each relation name of  ${f R}$  acts as unary predicate, holding *tuples*,
- attributes of tuples are accessed by path expressions variable.attrname,

#### Example

Names of all countries that have a city with more than 1,000,000 inhabitants:

```
{x.name | Country(x) \land \exists y : (City(y) \land y.country = x.code \land y.population > 1,000,000)}
```

• The Tuple Relational Calculus is a "parent" of SQL:

SELECT x.name	SELECT x.name
FROM country x, city y	FROM country x
WHERE y.country = x.code	WHERE EXISTS (SELECT *
AND y.population > 1000000	FROM city y
	WHERE y.country = $x.code$
	AND y.population > 1000000)

#### Examples

The following sets specify answers to sample queries:

- Names of all countries such that all its cities have more than 1,000,000 inhabitants:
  - $\{c.\mathsf{name} \mid \mathsf{Country}(c) \land \forall y : ((\mathsf{City}(y) \land y.\mathsf{country} = c.\mathsf{code}) \rightarrow y.\mathsf{population} > 1000000) \}$
- Names of all countries such that the capital of the country has more than 1,000,000 inhabitants:

 $\{c.\mathsf{name} \mid \mathsf{Country}(c) \land \\$ 

 $\exists y : (City(y) \land c.capital = y.name \land c.code = y.country \land c.capprov = y.province \land y.population > 1000000) \}$ 

 Names of all countries such that the country is a member of the organization with name "Europ.Union":

```
\{c.\mathsf{name} \mid \mathsf{Country}(c) \land \exists o, m : (\mathsf{Organization}(o) \land o.\mathsf{name} = \mathsf{``Europ}.\mathsf{Union''} \land d.\mathsf{name} = \mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf{``Europ}.\mathsf
```

```
m.country = c.code \land m.organization = o.abbrev) }
```

446

# 8.4 Formulas as Queries

Formulas can be seen as queries against a given database state:

- For a formula *F* with free variables  $X_1, \ldots, X_n, n \ge 1$ , write  $F(X_1, \ldots, X_n)$ .
- each formula  $F(X_1, ..., X_n)$  defines dependent on a given interpretation S an answer relation  $S(F(X_1, ..., X_n))$ .

The **answer set** to  $F(X_1, \ldots, X_n)$  wrt. S is the set of tuples  $(a_1, \ldots, a_n)$ ,  $a_i \in D$ ,  $1 \le i \le n$ , such that F is true in S when assigning each of the variables  $X_i$  to the constant  $a_i$ ,  $1 \le i \le n$ .

Formally:

 $S(F) = \{\{\beta(X_1), \dots, \beta(X_n)\} \mid S \models_{\beta} F \text{ where } \beta \text{ is a variable assignment of } free(F)\}.$ Each  $\beta$  such that  $S \models_{\beta} F$  is called an **answer**.

for n = 0, the answer to F is true if S ⊨<sub>∅</sub> F for the empty variable assignment Ø;
 the answer to F is false if S ⊭<sub>∅</sub> F for the empty variable assignment Ø.



448

#### Example 8.7

Consider the MONDIAL schema.

• Which cities (CName, Country) have at least 1,000,000 inhabitants?

 $F(CN,C) = \exists Pr, Pop, L_1, L_2 : (\textit{city}(CN,C,Pr,Pop,L_1,L_2) \land Pop \ge 1000000)$ 

The answer set is

 $\{\{CN/``Berlin", C/``D"\}, \{CN/``Munich", C/``D"\}, \{CN/``Hamburg", C/``D"\}, \}$ 

 $\{CN/$  "Paris", C/ "F"},  $\{CN/$  "London", C/ "GB"},  $\{CN/$  "Birmingham", C/ "GB"}, ... \}.

• Which countries (CName) belong to Europe?

$$\begin{split} F(CName) &= \exists \ CCode, Cap, Capprov, Pop, A, ContName, ContArea, Perc: \\ & (\textit{country}(CName, CCode, Cap, Capprov, Pop, A) \land \\ & \textit{continent}(ContName, ContArea) \land \\ & ContName = \texttt{``Europe''} \land \textit{encompasses}(CCode, ContName, Perc) ) \end{split}$$

## **CONJUNCTIVE QUERIES**

... the above ones are *conjunctive queries*:

- use only logical conjunction of positive literals
   (i.e., no disjunction, universal quantification, negation)
- conjunctive queries play an important role in database optimization and research.
- in SQL: only a single simple SFW clause without subqueries.

450

#### Example 8.7 (Continued)

• Again, relational division ... Which organizations have at least one member on each continent

 $F(Abbrev) = \exists O, HeadqN, HeadqC, HeadqP, Est:$ 

 $(organization(O, Abbrev, HeadqN, HeadqC, HeadqP, Est) \land$ 

 $\forall Cont: \ ((\exists ContArea: \ continent(Cont, ContArea)) \rightarrow \\$ 

 $\exists Country, Perc, Type : (encompasses(Country, Cont, Perc) \land$ 

isMember(Country, Abbrev, Type)))))

Negation

All pairs (country,organization) such that the country is a member in the organization, and all its neighbors are not.

$$\begin{split} F(CCode, Org) &= \exists CName, Cap, Capprov, Pop, Area, Type : \\ & (country(CName, CCode, Cap, Capprov, Pop, Area) \land \\ & isMember(CCode, Org, Type) \land \\ & \forall CCode' : (\exists Length : sym_borders(CCode, CCode', Length) \rightarrow \\ & \neg \exists Type' : isMember(CCode', Org, Type'))) \end{split}$$

# 8.5 Comparison of the Algebra and the Calculus

#### Algebra:

- The semantics is given by evaluating an algebraic expression (i.e., an operator tree) "algebraic Semantics" (which is also some form of a *declarative* semantics).
- The algebraic semantics also induces a naive, but already polynomial bottom-up evaluation algorithm based on the algebra tree.

#### Calculus:

- The semantics (= answer) of a query in the relational calculus is defined via the truth value of a logical formula wrt. an interpretation
   "logical Semantics" (which is some form of a *declarative* semantics)
- The logical semantics can be evaluated by a (FOL) Reasoner FOL is undecidable.
- $\Rightarrow$  translate "FOL" formulas over a simple database into the algebra ...

#### Example: Expressing Algebra Operations in the Calculus

Consider relation schemata R[A, B], S[B, C], and T[A].

(Note: [A, B] is the *format* of the relationships wrt. the relational model with named columns; *X* and *Y* are variables used in the *positional* relational calculus)

<b>Projection</b> $\pi[A](R)$ :	$F(X) = \exists Y R(X, Y)$
Selection $\sigma[A = B](R)$ :	$F(X,Y) = R(X,Y) \land X = Y$
Join $R \bowtie S$ :	$F(X,Y,Z) = R(X,Y) \wedge S(Y,Z)$
Union $R \cup (T \times \{b\})$ :	$F(X,Y) = R(X,Y) \lor (T(X) \land Y = b)$
<b>Difference</b> $R - (T \times \{B : b\})$ :	$F(X,Y) = R(X,Y) \land \neg(T(X) \land Y = b)$
<b>Division</b> $R \div T$ :	$F(Y) = (\exists X : R(X,Y)) \land \forall X : (T(X) \to R(X,Y))  \text{or}$
	$F(Y) = (\exists X : K(X, Y)) \land \neg \exists X : (I(X) \land \neg R(X, Y))$

## SAFETY AND DOMAIN-INDEPENDENCE

- For some formulas, the actual answer set does not depend on the actual database state, but on the domain of the interpretation.
- If the domain is infinite, the answer relations to some expressions of the calculus can be infinite!

### Example 8.8

Recall S = (I, D), usually D = Strings + Numbers + Dates (cf. Slide 443).

• Consider  $F(X) = \neg R(X)$  ("all a such that R(a) does not hold") where  $I(R) = \{(1)\}$ .

For every domain D, the answers to S(F) are all elements of the domain. For an infinite domain, e.g.,  $D = \mathbb{N}$ , the set of answers is infinite.

- Consider F(X,Z) = ∃Y(R(X,Y) ∨ S(Y,Z)), where I(R) = {(1,2)}, arbitrary S(S) (even empty). How to determine Z? - return {X/1, Y/d} for every element d of the domain?
- Consider  $F(X) = \forall Y : R(X, Y)$ where  $I(R) = \{(1, 1), (1, 2)\}$ . For  $\mathcal{D} = \{1, 2\}$  the answer set is  $\{\{X/1\}\}$ , for any larger domain, the answer set is empty.

454

#### Example 8.9

Consider a FOL interpretation S = (I, D) of persons:

Signature  $\Sigma = \{ married/2 \}$ , married(X, Y): X is married with Y.

 $F(X) = \neg married(john, X) \land \neg (X = john).$ 

#### What is the answer?

- Consider  $\mathcal{D} = \{john, mary\}, I(married) = \{(john, mary), (mary, john)\}.$  $\mathcal{S}(F) = \emptyset.$ 
  - there is no person (except John) who is not married with John
  - all persons are married with John???
- Consider  $\mathcal{D} = \{john, mary, sue\}$ ,  $I(married) = \{(john, mary), (mary, john)\}$ .  $\mathcal{S}(F) = \{\{X/sue\}\}$ .

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 S = (I, D), ADOM(F) contains

- all constants in F,
- and all constants in I(R) where R is a relation name that occurs in F.

 $ADOM(F \cup I)$  is called the **active domain** domain of F wrt. the interpretation I.

 $ADOM(F \cup I)$  is finite.

456

#### Domain-Independence

Formulas in the relational calculus are required to be domain-independent:

#### **Definition 8.2**

A formula  $F(X_1, ..., X_n)$  is **domain-independent** if for all interpretations I of the predicates and constants, and for all  $\mathcal{D} \supseteq ADOM := ADOM(F \cup I)$ ,

$$(I, ADOM)(F) = = \{(\beta(X_1), \dots, \beta(X_n)) \mid (I, ADOM) \models_{\beta} F, \ \beta(X_i) \in ADOM \text{ for all } 1 \le i \le n\} \\= \{(\beta(X_1), \dots, \beta(X_n)) \mid (I, \mathcal{D}) \models_{\beta} F, \ \beta(X_i) \in \mathcal{D} \text{ for all } 1 \le i \le n\} = (I, \mathcal{D})(F).$$

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

Idea: every formula guarantees that variables can only be bound to values from the database or that occur in the formula.

#### Safety: SRNF

#### **Definition 8.3**

A formula *F* is in **SRNF (Safe Range Normal Form)** [Abiteboul, Hull, Vianu: Foundations of Databases] if and only if it satisfies the following conditions:

- variable renaming: no variable symbol is bound twice with different scopes by different quantifiers; no variable symbol occurs both free and bound.
- remove universal quantifiers by replacing  $\forall X : G$  by  $\neg \exists X : \neg G$ ,
- remove implication by replacing  $F \to G$  by  $\neg F \lor G$ ,
- push negations down through ∧ and ∨.
   Negated formulas are then either of the form ¬∃F or ¬atom (push negations down through ∧ and ∨),
- flatten  $\land$ ,  $\lor$  and  $\exists$  (i.e., replace  $F \land (G \land H)$  by  $F \land G \land H$ , and  $\exists X : \exists Y : F$  by  $\exists X, Y : F$ ).

... then, check, whether the formula is *safe-range* as defined on the next slide:

458

#### Safety Check for SRNF formulas

#### **Definition 8.4 (Range-Restricted Variables, Safe-Range Formulas)**

- 1. For a formula F in SRNF, rr(F) is defined (and computable) via structural induction:
  - (1)  $F = R(t_1, \ldots, t_n) \implies rr(F)$  is the set of variables occurring in  $t_1, \ldots, t_n$

(2) 
$$F = x = a \text{ or } a = b \Rightarrow rr(F) = \{x\}$$

(3) 
$$F = F_1 \wedge F_2 \qquad \Rightarrow rr(F) = rr(F_1) \cup rr(F_2)$$

(4) 
$$F = F_1 \land X = Y \qquad \Rightarrow \begin{cases} rr(F) = rr(F_1) \cup \{x, y\} & \text{if } rr(F_1) \cap \{x, y\} \neq \emptyset \\ rr(F) = rr(F_1) & \text{if } rr(F_1) \cap \{x, y\} = \emptyset \end{cases}$$

(5) 
$$F = F_1 \lor F_2 \qquad \Rightarrow rr(F) = rr(F_1) \cap rr(F_2)$$

(6) 
$$F = \neg F_1$$
  $\Rightarrow$   $rr(F) = \emptyset$   
(7)  $F = \exists \bar{X} : F_1$   $\Rightarrow$   $\begin{cases} rr(F) = rr(F_1) - \bar{X} & \text{if } \bar{X} \subseteq rr(F_1) \\ \text{return } \bot & \text{if } \bar{X} \not\subseteq rr(F_1) \end{cases}$ 

2. if free(F) = rr(F) and no subformula returned  $\bot$ , F is safe range.

Note:

- \* The  $\forall$ -quantifier is not allowed in any formula in SRNF (i.e. replace  $\forall XF$  by  $\neg \exists X \neg F$ ).
- \* The definition does not contain any explicit syntactical hints how to write such a formula.

П

# Example 8.10 and Exercise

Consider the formulas

1.  $F(X, Y, Z) = p(X, Y) \land (q(Y) \lor r(Z)),$ 2.  $F(X, Y) = p(X, Y) \land (q(Y) \lor r(X)),$ 

3.  $F(X) = p(X) \wedge \exists Y : (q(Y) \wedge \neg r(X, Y)),$ 

4.  $F(X) = p(X) \land \neg \exists Y : (q(Y) \land \neg r(X, Y))$  – the relational division pattern,

5.  $F(X,Y) = p(X,Y) \land \neg \exists Z : r(Y,Z)$ ,

Are they in SRNF, and are they safe-range?

Give rr(G) for each of their subformulas.

Translate the formulas into SQL and into the relational algebra.

460

#### Safe Range and Domain Independence

#### Theorem 8.1

If a formula F is in SRNF and is safe-range, then it is domain-independent.

... one can prove this by induction, but this will also follow in a more useful way.

How to evaluate calculus queries?

- the underlying framework is FOL, undecidable, no complete reasoners exist.
   incomplete reasoners would do it, but they have high complexity and bad performance.
   (this issue will be the same when continuing with Datalog "knowledge" bases.)
- the goal is that the relational calculus is equivalent with the relational algebra; i.e. much weaker than full FOL, but polynomial.

(Datalog variants are also weaker than FOL, but some of them harder than polynomial)

 $\Rightarrow$  get a translation to the relational algebra.

(this problem will be solved by algebra+fixpoint and Logic-Programming-based implementations)

#### Comments on SRNF

- underlying idea: the formula can be evaluated from the database relations, never using the (purely logical concept of) "domain".
- subformulas of a conjunction *F*(..., *X*,...) ∧ *G*(*X*, *Y*) whose evaluation would not be domain-independent alone (i.e., *rr*(*G*) ⊊ *free*(*G*)) are "cured" by other parts of the conjunction (cf. solution to Example 8.10);
  - cf. correlated subqueries (SQL) or correlated joins in SQL/OQL/XQuery;
  - cf. index-based join in SQL: compute  $E_1 \bowtie E_2$  by iterating over results of  $E_1$  and accessing matching tuples in  $E_2$  via index.
  - also called "sideways information passing strategy".
- ... but the relational algebra does not have correlated subqueries (no subqueries in selection conditions at all!) and no correlated joins.

The algebra's theory is only bottom-up (cf. the relational algebra translations from Example 8.10 which provide some insights into the next definition ...).

462

#### Self-Containedness of Subformulas

#### **Definition 8.5**

A formula *F* that is in SRNF and which is safe-range is in **RANF (Relational Algebra Normal Form)** if:

- 1. (from SRNF) F does not contain  $\forall$  quantifiers (replace  $\forall XG$  by  $\neg \exists X \neg G$ ),
- *2.* (from SRNF) negated formulas are either of the form  $\neg \exists F$  or  $\neg$  atom (push negations down through  $\land$  and  $\lor$ ),
- 3. and if each subformula G of F is self-contained, where a subformula G is self-contained if
  - (0) if G is an atom, or if  $G = G_1 \land \ldots \land G_k$ (in this case, no additional explicit condition is stated, but requirements are made whenever such a G is used as a subformula in (i)-(iii)),
  - (i) if  $G = H_1 \lor \ldots \lor H_k$  and for all  $i, rr(H_i) = free(G)$ (which implies that  $free(H_i) = free(G) = rr(H_i)$  for all i),
  - (ii) if  $G = \exists \overline{X} : H$  and rr(H) = free(H)(which due to SRNF(7) is equivalent to rr(G) = free(G)),

(iii) if 
$$G = \neg H$$
 and  $rr(H) = free(H)$ .

(note: typo in [Abiteboul, Hull, Vianu: Foundations of Databases] in (ii) and (iii)!)

#### Self-Containedness of Subformulas

- Recall "correlated joins/subqueries" via  $F(...,X,...) \wedge G(X,Y)$  that refer to an "outer" query that provides bindings for –in this case– X.
- self-containedness requires that the evaluation of *G* does actually *not* depend on propagation of bindings from "outside".
- · For that,

$$rr(G) = free(G)$$
 (\*)

would be a sufficient criterion

(i.e., each subformula G is in SRNF itself).

This criterion is enforceable, except for negated subformulas.

464

#### Self-Containedness

Consider again

$$rr(F) = free(F)$$
 (\*)

• The definition of "self-contained" does not state any explicit condition on conjunctions  $G = G_1 \land \ldots \land G_k$ .

For them, the property (\*) follows from the other requirements:

if *G* is in a disjunction (from (3a)), in a negated subformula (from (3b)), and in an existence formula (from (3c) and SRNF (1.7)), and if G = F, then from SRNF (2).

- Self-containedness implies and requires that (\*) holds for all formulas that are not of the form  $F = \neg G$ .
- For negations  $F = \neg G$ ,  $rr(F) = \emptyset$ , and (\*) is implied and required only for their body: rr(G) = free(G).

Negations as a whole and isolated cannot satisfy (\*) – they depend on propagation from outside.

• idea: hardcode the subformula that generates the relevant bindings into the subformula.

#### From SRNF to RANF

Application of the following *rewriting rules (recursively – top-down)* translates safe-range SRNF formulas to RANF.

[Abiteboul, Hull, Vianu: Foundations of Databases]

- 1. Assume that (\*) holds for the whole formula F: free(F) = rr(F).
- 2. This is the case for each safe-range SRNF formula, so the starting point is well-defined.
- 3. input to each rewriting rule is a conjunction F of the form  $F = F_1 \land \ldots \land F_n$  s.t. free(F) = rr(F) where one or more of the  $F_i$  are not self-contained (let m the number of such  $F_i$ ).
- $\Rightarrow$  Make them self-contained!
- 4. each application of a rewriting rule will handle one such conjunct.
- 5. after *m* applications, *F* has been transformed into a conjunction  $F' = F'_1 \land \ldots \land F'_k$ ,  $k \le n$ , where all  $F'_i$  are self-contained.
- 6. then, the assumption in (\*) is valid for them (for negations: for their immediate subformula), and the formulas on lower levels can be rewritten.
- 7. as seen above, rewriting rules must only care for conjunctions (where the bindings propagation takes place).

#### From SRNF to RANF -2-

- W.I.o.g. assume that the conjunct to be treated is the rightmost one.
- Push-into-or:  $F = F_1 \land \ldots \land F_n \land G$  where  $G = G_1, \ldots, G_m$  is a disjunction, G is not self-contained, i.e.,  $rr(G) \subsetneq free(G)$  (which actually is the case if for some disjunct  $rr(G_i) \subsetneq free(G)$ ).

(w.l.o.g., G is the last conjunct)

Known: rr(F) = free(F); the missing variable(s) must be in  $rr(F_1, \ldots, F_n)$ .

Choose any subset  $F_{i_1}, \ldots, F_{i_k}$ ,  $k \leq n$  such that

 $G' = (F_{i_1} \land \ldots \land F_{i_k} \land G_1) \lor \ldots \lor (F_{i_1} \land \ldots \land F_{i_k} \land G_m)$  satisfies rr(G') = free(G').

- choosing all  $F_i$  is correct, but usually "inefficient".
- note:  $rr(G') \supseteq rr(G)$  ("=" in the best case), and for each disjunct  $G'_i$  in G',  $rr(G'_i) = free(G'_i) = free(G')$  (before,  $free(G_i) \neq free(G_j)$  was possible)

Let  $j_1, \ldots, j_{n-k}$  the indexes from  $\{1, \ldots, n\} \setminus \{i_1, \ldots, i_k\}$ ; i.e., the non-chosen ones. Replace *F* by  $F' = SRNF(F_{j_1} \land \ldots \land F_{j_{n-k}} \land G')$  and go on recursively.  $(SRNF(\ ))$  for renaming vars, flattening, etc.)

• ... two more rewriting rules see next slide.

#### From SRNF to RANF -3-

#### Example 8.11

- Recall Example 8.10 (2) and its algebra translation.
- Recall Example 8.10 (3) for guessing the next rule.
- ... recall Example 8.10 (4) for guessing the third rule.

... other rewriting rules in the same style:

- Push-into-exists:  $F = F_1 \land \ldots \land F_n \land \exists \overline{X} : G$  where rr(F) = free(F);  $rr(G) \subsetneq free(G)$ . Choose again  $F_i$ s such that  $G' = F_{i_1} \land \ldots \land F_{i_k} \land G$  as above. Replace F by  $F' = SRNF(F_{j_1} \land \ldots \land F_{j_{n-k}} \land \exists x : G')$  and go on recursively.
- Push-into-not-exists:  $F = F_1 \land \ldots \land F_n \land \neg \exists \overline{X} : G$  where rr(F) = free(F);  $rr(G) \subsetneq free(G)$ .

Do the same as above for  $G' = F_{i_1} \land \ldots \land F_{i_k} \land G$ , replace F by  $F' = SRNF(F_1 \land \ldots \land F_n \land \neg \exists x : G')$  (keeping all  $F_i$  also outside!) and go on recursively.

 what about "Push-into-negation"? Recall from Definition 8.5(2) that ¬ occurs only as ¬∃F (see above) or ¬*atom* (always self-contained).

468

#### Exercise

Consider the formula

```
F(X,Y) = \exists V : (r(V,X) \land \neg s(X,Y,V)) \land \exists W : (r(W,Y) \land \neg s(Y,X,W))
```

- Give rr(F) for all its subformulas,
- is it in SRNF?
- if yes, transform it to RANF.

This is an example, where no conjunct of the original formula is self-contained.

#### Exercise

Give an algorithm that transforms RANF formulas to the Relational Algebra.

#### PREVIEW

RANF is not only necessary for the translation into the Relational Algebra, but also for translation into (Nonrecursive Stratified) Datalog; cf. next section.

#### An Alternative Formulation

[Ullman, J. D., Principles of Database and Knowledge-Base Systems, Vol. 1]

#### **Definition 8.6**

A formula F is safe (SAFE) if:

- 1. *F* does not contain  $\forall$  quantifiers (replace  $\forall XG$  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  $F_1 \land \ldots \land F_m, m \ge 1$  of *F*:

All free variables must be limited, where limited is defined as follows:

• if  $F_i$  is neither a comparison, nor a negated formula, any free variable in  $F_i$  is limited,

- if  $F_i$  is of the form X = a or a = X with a a constant, then X is limited,
- if  $F_i$  is of the form X = Y or Y = X and Y is limited, then X is also limited.

(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).

#### Theorem 8.2

Safe formulas are domain-independent.

470

## Safety (Cont'd)

#### Example 8.12

- *p*(*X*, *Y*) ∨ *X* = *Y* is not safe: *X* = *Y* is a maximal conjunctive subformula where none of the variables is limited (it is also not domain-independent).
- $p(X,Y) \wedge X = Z$  is safe: p(X,Y) limits X and Y, then X = Z also limits Z.
- $p(X,Y) \land (q(X) \lor r(Y))$  is not safe, but the equivalent formula  $(p(X,Y) \land q(X)) \lor (p(X,Y) \land q(Y))$  is safe.
- $p(X, Y, Z) \land \neg(q(X, Y) \lor r(Y, Z))$  is not safe, but the logically equivalent formula  $p(X, Y, Z) \land \neg q(X, Y) \land \neg r(Y, Z)$  is safe.
- *F*(*X*) = *p*(*X*) ∧ ¬∃*Y* : (*q*(*Y*) ∧ ¬*s*(*X*, *Y*)) is not safe because *F*'(*X*) = ∃*Y* : (*q*(*Y*) ∧ ¬*r*(*X*, *Y*) is a maximal conjunctive subformula, but it does not limit *X*);
  the logically equivalent, but less intuitive formula *F*(*X*) = *p*(*X*) ∧ ¬∃*Y* : (*p*(*X*) ∧ *q*(*Y*) ∧ ¬*r*(*X*, *Y*)) is safe. (again the relational division pattern) □

#### Notes

- condition RANF(3b) is not required by SAFE. Nevertheless, since in  $\neg G$ , G is a maximal conjunctive formula (maybe with m = 1), SAFE(3) applies to it and implies RANF(3b).
- condition RANF(3a) is stronger than SAFE(2), but implied by SAFE(3) since in G<sub>1</sub> ∨ G<sub>2</sub> each disjunct is a maximal conjunctive subformula which implies that all its variables must be limited.
- SAFE(3) explicitly requires for each negated formula ¬F(X) that it *must* occur in some conjunction G = (... ∧ F(X) ∧ ...) with positive formulas that limit the Xs:
   Otherwise, if any non-conjunctive formula G contains ¬F(X) as an immediate subformula, ¬F(X) would be a maximal conjunctive formula in F where X are not limited.
- In contrast, RANF does not state an explicit condition on the occurrence of negated subformulas. Implicitly, the same condition follows from the fact that *rr*(¬*F*(*X*)) = Ø (SNRF(6)), and the remark on the bottom of Slide 463: *X* ⊂ *free*(*G*), so there must be a conjunct *G<sub>i</sub>* "neighboring" the negated formula to such that *rr*(*G<sub>i</sub>*) ⊆ *X*.

#### Safety: universal quantification

Consider again from Example 8.8:

$$F(X) = \forall Y : R(X, Y)$$

• This formula is not allowed to be considered since  $\forall$  must be rewritten:

$$F_2(X) = \neg \exists Y : \neg R(X, Y)$$

is not safe since  $\neg R(X, Y)$  is a maximal conjunctive subformula.

- Start again with *F*: the problem in Example 8.8 was that it is not known which *Y* have to be considered (the whole domain?)
- restrict to *Y* that satisfy some condition (e.g., all country codes).

An upper bound is to consider all elements of the active domain, let (assume relations  $R_{/2}, S_{/1}, \ldots$ )

$$ADOM(Z) = (\exists Y : R(Z, Y) \lor \exists X : R(X, Z) \lor S(Z) \lor \ldots) :$$
  
$$F_3(X) = \forall Y : (ADOM(Y) \to R(X, Y))$$

(continue next slide)

Safety: universal quantification (cont'd)

• ... and rewrite ∀:

 $F_4(X) = \neg \exists Y : \neg (ADOM(Y) \to R(X, Y))$ 

push negation down and rewrite  $F \rightarrow G$  as  $\neg F \lor G$ :

 $F_5(X) = \neg \exists Y : (ADOM(Y) \land \neg R(X, Y))$ 

•  $ADOM(Y) \land \neg R(X, Y)$  is still not safe. X must be bound; use again ADOM:

 $F_6(X) = \neg \exists Y : (ADOM(X) \land ADOM(Y) \land \neg R(X,Y))$ 

• is safe, but unintuitive. Pulling out X yields ...

$$F_7(X) = ADOM(X) \land \neg \exists Y : (ADOM(Y) \land \neg R(X, Y))$$

... which is the relational division pattern!

474

Aside: Another Alternative Formulation

[Allen Van Gelder and Rodney W. Topor. Safety and translation of relational calculus queries. ACM Transactions on Database Systems (TODS), 16(2):235-278, 1991.]

- based on two syntactical, inductively defined properties con(X) ("constrained") and gen(X) ("generated"),
- a formula is "evaluable" if
  - for every free variable in Q(X) = F(X), gen(X, F) holds,
  - for every subformula  $\exists X : F$ , con(X, F) holds,
  - for every subformula  $\forall X : F$ ,  $con(X, \neg F)$  holds,
- claimed that this definition is the largest class of domain-independent formulas that can be characterized by syntactical restrictions;
- proven that for queries without repetitions of predicate symbols the definition coincides with domain-independence.
  - The (simple) formula Q(x) = p(x) ∧ ∀y : ¬q(x, y) is in SRNF, and evaluable, but the equivalent PLNF (prenex literal normal form) Q'(x) = ∀y : (p(x) ∧ ¬q(x, y)) is not in SRNF (equivalent to ¬∃y : ¬(p(x) ∨ ¬q(x, y)), where y ∉ rr(¬(p(x) ∨ ¬q(x, y)))), but still "evaluable". Later, for Datalog always the (SRNF-compatible) variant where the scope of the universal quantifier is *only a single, negative literal* is relevant.

# SUMMARY: A HIGHER-LEVEL VIEW ON DOMAIN INDEPENDENCE/SAFETY VS RANF

#### **Domain Independence**

- Domain independence is absolutely necessary for a query to have a well-defined meaning (humans evaluate such queries when the context gives the domain, e.g. "who is not registered for the exam?" [domain: the participants of the lecture]).
- Domain independence is undecidable.

#### Safety

- · safety is defined purely syntactically,
- safety can be tested effectively,
- safety implies domain-independence.

476

# METALEVEL: RECONSIDER FOL VS HERBRAND STYLE

• FOL:

 $\Sigma$ : predicate symbols  $p, q, r, \ldots$ , function symbols  $f, g, \ldots$ , constant symbols  $a, b, c, \ldots$ ,

 $\mathcal{I} = (I, \mathcal{D}); \quad I(p) \subseteq \mathcal{D}^n \text{ for } n\text{-ary } p.$ 

 $\mathcal{I} \models p(a,b,c) \ \Leftrightarrow \ (I(a),I(b),I(c)) \in I(p).$ 

- The abstraction level of *I* is needed in FOL model theory, especially if function symbols are used.
- the notion of the domain  $\mathcal{D}$  is needed for the semantics of the universal quantifier and proving validity of a formula.
- Herbrand/DB with *safe formulas*:
  - $\Sigma$ : predicate symbols  $p, q, r, \ldots$ ,

constants  $a, b, c, \ldots$  + datatype values 1, 2, 3, ..., "D", "CH", ...

Database state S over the relations  $p, q, r, \ldots$ ;

with values from the constants and datatype values,

- $\mathcal{S}\models p(a,b,c) \ \Leftrightarrow \ (a,b,c)\in p.$
- $\Rightarrow$  neither need the notions of *I* nor  $\mathcal{D}$  everything is immediately contained in  $\mathcal{S}$ .

#### Domain Independence is inherent in the relational algebra and in SQL

#### Algebra

- Basic algebra expressions/leaves of the algebra tree are always relations (database relations or constants),
- (non-atomic) "negation" in the relation algebra only via "minus",
- proof by structural induction: the left subtree of "minus" is always domain-independent ⇒ the whole expression is domain-independent.

#### SQL

- FROM clause always refers (positively) to relations or to SQL subqueries,
- (non-atomic) negation only in subqueries in the WHERE clause, sideways-information-passing.
- whole SQL expression is domain-independent.

478

#### A Higher-Level View on Domain Independence/Safety vs RANF

- Logics: domain-independent formulas can be evaluated;
- Relational algebra: requires RANF for strict bottom-up evaluation;
- SQL:
  - relaxed criterion (cf. Example 8.10) for (negated) existential quantification;
  - not relaxed for disjunction/union;
  - ⇒ internal compiler from SQL into an internal (relational) algebra that supports sideways information passing;
- SPARQL (query language for RDF): also relaxed for disjunction/union.
- Datalog will require RANF since every subexpression is represented by an own "local" rule;

"global" semantics and internal compilation by Logic Programming-based (Prolog) top-down proof tree strategy supports *sideways information passing*.

# 8.6 Equivalence of Algebra and (safe) Calculus

As for the algebra, the attributes of each relation are assumed to be ordered.

#### Theorem 8.3

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 S, Q and F define the same answer relation.

#### **Proof Summary**

- give mappings (A) "Algebra  $\rightarrow$  Calculus" and (B) "Calculus  $\rightarrow$  Algebra"
- (A) gives insights how to express a textual (or SQL) query by Datalog Rules,
- (B) gives insight how to write SQL statements for a given textual (or logical) query (and how one could implement a Calculus evaluation engine via SQL).

480

## 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).

All generated formulas are safe.

As an invariant, the variable names  $A, B, C, \ldots$  correspond always to the column names A,B,C,... of the format of the respective algebra expression.

Induction base: Q does not contain operators.

• if Q = R where R is a relation symbol of arity  $n \ge 1$  with format  $A_1, \ldots, A_n$ :

$$F(A_1,\ldots,A_n)=R(A_1,\ldots,A_n)$$

R						
$A_1$	$A_2$					
а	1					
b	2					

	$A_1$	$A_2$
answer to $R(A_1, A_2)$ :	а	1
	b	2

• otherwise,  $Q = \{A:c\}$  where c is a constant. Then, F(A) = (A = c).

A:cAnswer to 
$$A = c$$
:AAC

## Induction step:

• Case 
$$Q = Q_1 \cup Q_2$$
. Thus,  $\Sigma_{Q_1} = \Sigma_{Q_2} = A_1, \dots, A_n$ .  
 $F(A_1, \dots, A_n) = F_1(A_1, \dots, A_n) \vee F_2(A_1, \dots, A_n)$   
Example:  

$$\begin{array}{c|c}
Q_1 \\
\hline
A_1 & A_2 \\
\hline
a & b \\
\hline
c & d
\end{array}$$
 $F_1( \begin{array}{c}
A_1 & A_2 \\
\hline
a & b \\
\hline
c & d
\end{array}$ 
 $F( \begin{array}{c}
A_1 & A_2 \\
\hline
a & b \\
\hline
c & d
\end{array}$ 
 $C & d \\
\hline
Q_2 \\
\hline
A_1 & A_2 \\
\hline
1 & 2 \\
\hline
c & d
\end{array}$ 
 $F_2( \begin{array}{c}
A_1 & A_2 \\
\hline
1 & 2 \\
\hline
c & d
\end{array}$ 
 $I = 2$ 

482

- Case  $Q = Q_1 Q_2$ . Analogously; replace  $\ldots \lor \ldots$  by  $(\ldots) \land \neg (\ldots)$ .
- Case  $Q = \pi[\bar{Y}](Q_1)$  with  $\bar{Y} = \{A_{i_1}, \dots, A_{i_k}\} \subseteq \Sigma_{Q_1}, k \ge 1$ . Let  $\{j_1, \dots, j_{n-k}\} = \{1, \dots, n\} \setminus \{i_1, \dots, i_k\}$  (the indices not in  $\bar{Y}$ ).

$$F(\bar{Y}) = F(A_{i_1}, \dots, A_{i_k}) = \exists A_{j_1}, \dots, A_{j_{n-k}} : F_1(A_1, \dots, A_n) .$$

Example:

$$\begin{array}{c|c} \hline Q_1 \\ \hline A_1 & A_2 \\ \hline A_2 \\ \hline C & d \\ \hline F(A_2) = \exists A_1 : F_1(A_1, A_2) \\ F(\underline{A_2}) \\ \hline B \\ \hline \end{array}$$

d

• Case  $Q = \sigma[\alpha](Q_1)$  where  $\alpha$  is a condition over  $\Sigma_{Q_1} = \{A_1, \dots, A_n\}$ .

$$F(A_1,\ldots,A_n)=F_1(A_1,\ldots,A_n)\wedge\alpha',$$

where  $\alpha'$  is obtained by replacing each column name  $A_i$  by the variable  $A_i$  in  $\sigma$ .

Examp	le:					
	$Q_1$			4		,
	$A_1$	$A_2$	$F_1($	<i>A</i> <sub>1</sub>	A <sub>2</sub>	
	1	2		3 I	2 1	
	3	4		0	-	
Let $\sigma$ =	= <b>"A</b> 1	= 3":	$F(A_1$	$(A_2)$	$= F_1$	$(A_1, A_2) \land A_1 = 3$
			F(	$A_1$	$A_2$	)
				3	4	

484

• Case  $Q = \rho[\mathsf{A}_1 \to \mathsf{B}_1, \dots, \mathsf{A}_m \to \mathsf{B}_m](Q_1), \Sigma_{Q_1} = \{\mathsf{A}_1, \dots, \mathsf{A}_n\}, n \ge m.$ 

 $F(B_1,\ldots,B_m,A_{m+1},\ldots,A_n) = \exists A_1,\ldots,A_m : (F_1(A_1,\ldots,A_n) \land B_1 = A_1 \ldots \land B_m = A_m)$ 

Example:

Consider  $\rho[A_1 \rightarrow B_1](Q_1)$ :

$$F(B_1, A_2) = \exists A_1 : (F_1(A_1, A_2) \land A_1 = B_1)$$

$$F( \begin{array}{c} B_1 & A_2 \\ \hline 1 & 2 \\ \hline 3 & 4 \end{array})$$

• Case  $Q = Q_1 \bowtie Q_2$  and  $\Sigma_{Q_1} = \{A_1, \dots, A_n\}, \Sigma_{Q_2} = \{A_1, \dots, A_k, B_{k+1}, \dots, B_m, \}, n, m \ge 1 \text{ and } 0 \le k \le n, m.$ 

 $F(A_1, \ldots, A_n, B_{k+1}, \ldots, B_m) = F_1(A_1, \ldots, A_n) \wedge F_2(A_1, \ldots, A_k, B_{k+1}, \ldots, B_k)$ 

Example:

	Q	$P_1$	$Q_2$					``	<b>F</b> (	4	D	,
	$A_1$	$A_2$	$A_1$	$B_2$	$F_1($	A <sub>1</sub>	A <sub>2</sub>	)	$F_2($	<u>A</u> 1	<i>B</i> <sub>2</sub>	)
1	1	2	5	6		у 1	2			ວ 1	0 7	
	3	4	1	7	1	0	4			I	1	

 $F(A_1, A_2, B_2) = F_1(A_1, A_2) \wedge F_2(A_1, B_2)$ F( A\_1 A\_2 B\_2 )

• Note that in all cases, the resulting formulas *F* are domain-independent, in SRNF, RANF, and SAFE.

(which came up automatically, because it is built-in in the structure induced by the algebra expressions)

#### (B) Calculus to Algebra

Consider a relational schema  $\Sigma = \{R_1, \ldots, R_n\}$  and a SAFE formula  $F(X_1, \ldots, X_n)$ ,  $n \ge 1$  of the relational calculus.

First, an algebra expression ADOM that computes the active domain ADOM(S) of the database state is derived:

For every  $R_i$  with arity  $k_i$ ,

 $ADOM(R_i) = \pi[\$1](R_i) \cup \ldots \cup \pi[\$k_i](R_i).$ 

(where  $\pi[\$i]$  denotes the projection to the *i*-th column). Let

 $ADOM = ADOM(R_1) \cup \ldots \cup ADOM(R_n) \cup \{a_1, \ldots, a_m\},\$ 

where  $a_1, \ldots, a_m$  are the constants occurring in *F*.

• For a given database state S over  $\Sigma$ , ADOM(S) is a unary relation that contains the whole active domain of the database, i.e., all values occurring in any tuple in any position.

An equivalent algebra expression Q is now constructed by induction over the number of maximal conjunctive subformulas of F.

**Induction base:** *F* is a conjunction of positive literals. Thus,  $F = G_1 \land \ldots \land G_l$ ,  $l \ge 1$ .

(1) Case l = 1. *F* is a single positive safe literal.

Then, either is of the form  $F = R_i(a_1, \ldots, a_{i_k})$ , where each  $a_j$  is a variable or a constant, or F is a comparison of one of the forms F = (X = c) or F = (c = X), where X is a variable and c is a constant (note that all other comparisons would not be safe).

- Case  $F = R(a_1, ..., a_{i_k})$ : contains some (free, maybe duplicate) variables, and some constants that state a condition on the matching tuples.
  - $\Rightarrow$  encode the condition into a selection, and do a projection to the columns where variables occur one column for each variable and name the columns with the variables:

e.g. 
$$F(X,Y) = R(a, X, b, Y, a, X)$$
. Then, let

$$Q(F) = \rho[\$2 \to X, \$4 \to Y](\pi[\$2, \$4](\sigma[\Theta_1 \land \Theta_2](R)))$$

where  $\Theta_1 = (\$1 = a \land \$3 = b \land \$5 = a)$  and  $\Theta_2 = (\$2 = \$6)$ .

- Case F = (X = c) or F = (c = X). Let  $Q(F) = \{X : c\}$ 

488

(2) Case l > 1 (cf. example below) Then, w.l.o.g.

 $F = G_1 \wedge \ldots \wedge G_m \wedge G_{m+1} \wedge \ldots \wedge G_l$ 

s.t.  $1 < m \le l$ , where all  $G_i$ ,  $1 \le i \le m$  as in (1) and all  $G_j$ ,  $m + 1 \le j \le l$  are other comparisons (i.e., unsafe literals like X = Y, X < 3).

For every  $G_i$ ,  $1 \le i \le m$  take an algebra expression  $Q(G_i)$  as done in (1). The format  $\Sigma_{Q(G_i)}$  is the set of free variables in  $G_i$ . Let

$$Q' = \bowtie_{i=1}^m Q(G_i).$$

With  $\Theta$  the conjunction of the additional conditions  $G_{m+1}, \ldots, G_l$ ,

$$Q(F) = \sigma[\Theta](Q')$$
.

#### Example 8.13

Consider  $F = R(a, X, b, Y, a, X) \land S(X, Z, a) \land X = Y \land Z < 3$ as  $F = G_1 \land G_2 \land G_3 \land G_4$ :

$$Q(G_1) = \rho[\$2 \to X, \$4 \to Y](\pi[\$2, \$4](\sigma[\$1 = a \land \$3 = b \land \$5 = a \land \$2 = \$6](R)))$$
  

$$Q(G_2) = \rho[\$1 \to X, \$2 \to Z](\pi[\$1, \$2](\sigma[\$3 = a](S)))$$
  

$$Q(F) = \sigma[X = Y \land Z < 3](Q(G_1) \bowtie Q(G_2))$$

**Structural Induction Step:** For formulas  $G, G_1, \ldots, G_l, H$  the equivalent algebra expressions are  $Q(G), Q(G_1), \ldots, Q(G_l), Q(H), \ldots$ 

(3)  $F = G \lor H$ :

 $Q(F) = Q(G) \cup Q(H)$ 

(safety guarantees that G and H have the same free variables, thus, Q(G) and Q(H) have the same format).

(4)  $F = \exists X : G$ :

 $Q(F) = \pi[\operatorname{Vars}(Q(G)) \setminus \{X\}](Q(G)) ,$ 

(5)  $F = \neg G$ , where Q(G) has columns/variables  $X_1, \ldots, X_k$ :

$$Q(F) = \rho[\$1 \to X_1, \dots, \$k \to X_k](ADOM^k) - Q(G)$$

(6) F = G<sub>1</sub> ∧ ... ∧ G<sub>l</sub>, l ≥ 2 is a maximal conjunctive subformula (difference to (2): now it's the induction step where the conjuncts are allowed to be complex subformulas):
 Q(F) is then constructed analogously to (2) as a join.

#### Understanding the Proof: Negation as Minus

The  $ADOM^k$  in "calculus to algebra" item (5) looks awkward. What is it good for? What does it *mean*?

- according to Def. 8.3 (4) (max. conjunctive subformulas), all the variables  $X_1, \ldots, X_k$  in a negative conjunct  $\neg G$  must occur positively in some other conjunct (and be bound by this).
- ⇒ instead of  $ADOM^k$ , the cartesian product (or any overestimate of it) of the possible values of  $X_1, \ldots, X_k$  can be used.
  - Formal example next slide,
  - practical MONDIAL example second next slide.

Understanding the Proof: Negation as Minus

#### Formal Example

$$F(X,Y) = p(X,Y,Z) \land \neg \exists V : q(Y,Z,V) .$$

•  $F_1(X,Y,Z) = p(X,Y,Z) \Rightarrow E_1 = \rho[\$1 \rightarrow X,\$2 \rightarrow Y,\$3 \rightarrow Z](p),$ 

- $F_2(Y, Z, V) = q(Y, Z, V) \quad \Rightarrow \quad E_2 = \rho[\$1 \rightarrow Y, \$2 \rightarrow Z, \$3 \rightarrow V](q),$
- $F_3(Y,Z) = \exists V : F_2(Y,Z,V) \Rightarrow E_3 = \pi[Y,Z](E_2) = \pi[Y,Z](\rho[\$1 \rightarrow Y,\$2 \rightarrow Z,\$3 \rightarrow V](q)),$
- $F_4(Y,Z) = \neg F_3(Y,Z) \Rightarrow \rho[\$1 \rightarrow Y, \$2 \rightarrow Z](ADOM^2) E_3 = \rho[\$1 \rightarrow Y, \$2 \rightarrow Z](ADOM^2) \pi[Y, Z](\rho[\$1 \rightarrow Y, \$2 \rightarrow Z, \$3 \rightarrow V](q))$ (yields all possible  $(y, z) \in ADOM^2$  that are not in ...)
- $F_5(X,Y,Z) = F_1 \wedge F_4 \implies E_1 \bowtie E_4 =$  $E_1 \bowtie (\rho[\$1 \rightarrow Y,\$2 \rightarrow Z](ADOM^2) - \pi[Y,Z](\rho[\$1 \rightarrow Y,\$2 \rightarrow Z,\$3 \rightarrow V](q)))$

Only pairs (Y, Z) can survive the join that are in the result of the first component. Thus, instead taking the "overestimate"  $ADOM^2$ ,  $\pi[Y, Z](E_1)$  can be used:

 $E_1 \bowtie (\pi[Y, Z](E_1) - \pi[Y, Z](\rho[\$1 \rightarrow Y, \$2 \rightarrow Z, \$3 \rightarrow V](q))).$ 

#### Negation as Minus - A practical example

• Ever seen this ADOM construct in exercises to the relational algebra? - No. Why not?

Consider relations country(name,country) and city(name,country,population):

 $F(CN, C) = \text{country}(CN, C) \land \neg \exists Cty, Pop : (\text{city}(Cty, C, Pop) \land Pop > 1000000)$ Structural generation of an equivalent algebra expression:

- $F_1(CN,C) = \operatorname{country}(CN,C) \quad \Rightarrow \quad E_1 = \rho[\$1 \to CN,\$2 \to C](\operatorname{country}),$
- $F_2(Cty, C, Pop) = \operatorname{city}(Cty, C, Pop) \land Pop > 1000000$  $\Rightarrow \quad E_2 = \rho[\$1 \rightarrow Cty, \$2 \rightarrow C, \$3 \rightarrow Pop](\sigma[\$3 > 1000000](\operatorname{city})),$
- $F_3(C) = \exists Cty, Pop : F_2(Cty, C, Pop)$  $\Rightarrow \quad E_3 = \pi[C](\rho[\$1 \rightarrow Cty, \$2 \rightarrow C, \$3 \rightarrow Pop](\sigma[\$3 > 1000000](city))),$
- $F_4(C) = \neg F_3(C) \Rightarrow E_4 = \rho[\$1 \to C](ADOM) E_3$  (abbreviating  $\pi(\rho(...))$  in  $E_3$ ) =  $\rho[\$1 \to C](ADOM) - \pi[\$2 \to C](\sigma[\$3 > 1000000](city))$

(yields all possible *C* that are not in ...)

At this point, one knows that not the complete ADOM (all values anywhere in the database) has to be considered, but that it is sufficient to consider all countrycodes:  $E'_4 = \pi[\$2 \rightarrow C](country) - \pi[\$2 \rightarrow C](\sigma[\$3 > 100000](city))$ 

And now, both parts of the outer conjunction are combined by a join:

$$\begin{split} F(CN,C) &= F_1(CN,C) \wedge F_4(C) \\ \Rightarrow & E_1 \bowtie E'_4 = \\ & \rho[\$1 \rightarrow CN,\$2 \rightarrow C](\text{country}) \bowtie (\pi[\$2 \rightarrow C](country) - \pi[\$2 \rightarrow C](\sigma[\$3 > 1000000](\text{city}))) \end{split}$$

494

# 8.7 Symbolic Reasoning

- Logics in general, and FOL are mathematical concepts. Research mathematically investigates different logics and their properties.
- Symbolic Reasoning applies logic-based algorithms on concrete problems, e.g.,
  - Software and hardware verification (e.g., correctness of automobile or airplane systems)
  - Answering queries against knowledge bases
- algorithms must operate on the syntax level:
  - formulas (i.e., parse-trees of formulas)
  - terms (i.e., parse-trees of terms)
  - sets of variable bindings
    - \* term unification,
    - \* answer bindings (to unification/matching and to queries)

# **DATALOG: HERBRAND SEMANTICS**

Logic programming (LP) frameworks (e.g., Prolog and Datalog) use the *Herbrand Semantics* (after the French logician Jacques Herbrand):

- a Herbrand Interpretation H = (H, D<sub>Σ</sub>) for a given signature Σ uses always the Herbrand Universe D<sub>Σ</sub> that consists of all terms that can be constructed from the function symbols (incl. constants) in Σ: john, father(john), germany, capital(germany), berlin, ....
- $\Rightarrow$  "every term is interpreted by itself"
  - the relation names are the predicate symbols in Σ, and they are also "interpreted by themselves (as a relation)", i.e., H(encompasses) = encompasses.
  - the *Herbrand Base*  $\mathcal{HB}_{\Sigma}$  is the set of all *ground atoms* over elements of the Herbrand Universe and the predicate symbols of  $\Sigma$ .
- $\Rightarrow$  A Herbrand Interpretation is a (finite or infinite) subset of the Herbrand Base.
  - $\mathcal{H} \models hasAncestor(john, father(john))$  if (john, father(john))  $\in hasAncestor$ .
  - in contrast, in traditional FOL:  $(I, D) \models hasAncestor(john, father(john))$  if  $(I(john), I(father(I(john)))) \in I(hasAncestor)$ .
  - if function symbols are allowed, usually with equality predicate  $\approx$ , e.g., father(john)  $\approx$  jack.

496

#### Deductive Databases: Datalog

- the domain consists of constant symbols and datatype literals.
- an interpretation  $\mathcal{H}$  is explicitly seen as a *finite* set of ground atoms over the predicate symbols and the Herbrand Universe:

country(ger, "Germany", "D", berlin, 356910,83536115), encompasses(ger, eur, 100).

 $\mathcal{H} \models \mathsf{encompasses}(\mathsf{ger},\mathsf{eur},100) \text{ if and only if } (\mathsf{ger},\,\mathsf{eur},100) \in \mathcal{H}(\mathsf{encompasses})$ 

 $\text{ if and only if } \quad \mathsf{encompasses}(\mathsf{ger},\,\mathsf{eur},100) \in \mathcal{H} \;.$ 

- Unique Name Assumption (UNA): different symbols mean different things.
- Datalog restricts the allowed formulas (cf. Slides 557 ff.):
  - conjunctive queries,
  - Datalog knowledge bases consist of rules of the form *head* ← *body* (variants: positive nonrecursive, recursive, + negation in the body, + disjunction in the head)
- special semantics/model theories for each of the variants: minimal model, stratified model, well-founded model, stable models
  - each of them characterized as sets of ground atoms.

# SEMANTIC WEB: RDF, RDFS, AND OWL

- RDF data model (see also Slide 440)
  - unary and binary predicates over literal values and URIs (Object (identifier)s; classes and properties are also represented by URIs)
- RDFS (RDF Schema): adds second order flavour:
  - RDF triples can have properties or classes as subject and object,
  - then use predefined RDFS predicates:
  - capital rdfs:domain Country; rdfs:range City. capital rdfs:subPropertyOf hasCity
  - semantics can be encoded in FOL rule patterns:  $\forall x, y : \operatorname{capital}(x, y) \rightarrow \operatorname{Country}(x) \land \operatorname{City}(y)$  $\forall x, y : \operatorname{capital}(x, y) \rightarrow \operatorname{hasCity}(x, y)$
  - mapped to FOL model theory.
  - RDFS and "OWL Lite" (see next slide) can be mapped to positive recursive Datalog
    - $\Rightarrow$  polynomial
      - \* just positive rules: CWA and OWA semantics coincide

498

#### Semantic Web: RDF, RDFS, and OWL (cont'd)

- · OWL: additional specialized vocabulary for describing Description Logic concepts
- Second order predicates predicates about predicates: borders a owl:SymmetricProperty.
   hasChild rdfs:subPropertyOf hasDescendant
   hasDescendant a owl:TransitiveProperty.
   SymmetricProperty(borders)
   hasChild L hasDescendant
   TransitiveProperty(hasDescendant)
- many OWL (OWL Lite) constructs can be translated into FOL (and Datalog) rule patterns:  $\forall x, y : borders(x, y) \rightarrow borders(y, x).$   $\forall x, y : hasChild(x, y) \rightarrow hasDescendant(x, y).$  $\forall x, y, z : hasDescendant(x, y) \land hasDescendant(y, z) \rightarrow hasDescendant(x, z).$
- Queries about data against RDF(+RDFS+OWL Lite) knowledge bases: *algebraic evaluation, polynomial.*
- Queries against RDF+OWL DL knowledge base: reasoning, exponential.

# Chapter 9 Reasoning

Queries vs. Conclusions

- Query: (for which *X*) does something hold in a given database state?
- Conclusions: given some facts and some knowledge, does something hold?

#### **Knowledge Bases**

- general statements/logic formulas (cf. human knowledge and reasoning)
- often definitions: "adults are persons who are at least 18 years old",
  "parents are persons who have children", "every person is either male or female",
  "my uncles are the male siblings of my parents, and the husbands of the female siblings of my parents",
- can often, but not always, be represented as rules.
- Queries against knowledge bases are answered by *reasoning*, not by algebraic evaluation (although some reasoning can be implemented on an algebraic base).

500

# 9.1 Model Theory and Logical Entailment

For formalizing (and applying) reasoning, logical entailment,

 $\mathcal{F}\models\varphi$ 

is needed:

- $\mathcal{F}$ : a set of formulas, the specification
- $\varphi$ : a formula
- does *F* entail φ,
  i.e., assumed that *F* holds, can we conclude that φ also holds?
- e.g.  $\mathcal{F}$  the specification of the notion of "uncle", and a given database of persons:
  - does "Bob is an uncle of Alice" hold?
  - which persons (in the database) are uncles of Alice?
  - which persons (in the database!) are not uncles of Alice?
    - \* remark on closed world vs. open world

# DATABASES VS. KNOWLEDGE BASES



Then, the semantics is given by the model(s) of the database state and the rules.





AreaManager: intermediate group leaders in a department)

What can be "models" of this ontology? How do you represent them? Give an example.
#### Semantics: Set theory

• a class is a set of instances,

Employee={alice, bob, john, mary, tom, larry} and Manager={alice, bob, john, mary}, AreaManager={mary} and TopManager={alice, bob, john}, Dept={sales, production, management}

Constraints from subclasses:  $Manager = AreaManager \cup TopManager$   $Manager \subseteq Employee$   $AreaManager \subseteq Manager$  and  $TopManager \subseteq Manager$  (both redundant)

• an attribute is a set of pairs of (i) an instance and (ii) an element of a literal domain (constraint!)

name =

{(alice, "Alice"), (bob, "Bob"), (john, "John"), (mary, "Mary"), (tom, "Tom"), (larry, "Larry")} salary =

*{(alice, 70000), (bob, 60000), (john, 100000), (mary, 40000), (tom, 25000), (larry, 20000)}* analogously for department names.

#### Semantics: Set theory (Cont'd)

a relationship is a pair of instances, (or a set of *n*-tuples, in case of *n*-ary relationships) *works-for* ⊆ *Employee* × *Dept works-for* = {(alice, sales), (mary, sales), (larry, sales), (bob, production), (bob, sales), (tom, production), (john, management)}
manages ⊆ TopManager × Dept manages = {(alice, sales), (bob, production), (john, management)} subordinate ⊆ Employee × Manager subordinate = {(mary, alice), (bob, alice), (larry, mary), (larry, alice), (tom, bob)}
not so obvious: constraints coming from the cardinality specifications, e.g., *the set of top managers is a subset of the things that manage exactly one department*,

the set of employees is a subset of the things that work for at least one and at most three departments

(see later after the discussion of first-order logic)



#### Adequateness of (extended) ER Diagrams

An ontology should give a concise characterization of a domain and its constraints.

- · classes, key constraints
- subclasses (specialization/generalization, disjointness)
- ranges and domains of properties (e.g., that the domain of "manages" is not all employees but only the managers)
- cardinalities
- is manages  $\subset$  works-for ?
- is manages  $\cap$  works-for =  $\emptyset$  ?
- · subproperty constraints cannot be expressed
- further constraints (e.g., employees that work for a department are subordinate to the department's manager) cannot be expressed.

#### Exercise

• Discuss alternatives for the cardinalities for "subordinate".

## ALTERNATIVE SEMANTICS: RELATIONAL MODEL

Exercise: give a relational schema and the corresponding database state to the above ER diagram.

Relational Schema as an Ontology

- basically non-graphical, can be supported e.g. by dependency diagrams (cf. the Mondial documentation)
- no distinction between "classes" and "relationships"
- · key constraints, foreign key/referential constraints
- keys/foreign key allow to guess classes vs. relationships
- cardinality "1": functional dependencies (adding n:1-relationships into a table like department/manages country/capital)
- no general cardinality constraints
- sometimes: domain constraints (by foreign keys)
- no further (inter- and intra-relation) constraints

508

## EXAMPLE: AXIOMATIZATION OF THE "COMPANY" ONTOLOGY

Consider again the ER diagram from Slide 503.

- give the *first-order signature*  $\Sigma$  of the ontology,
- · formalize the constraints given in the
  - subclass constraints
  - range and domain constraints
  - cardinality constraints

and

- additional constraints/definitions that cannot be expressed by the ER model.
- (this set of formulas is called a first-order "theory" or "axiomatization" of the ontology)
- express the instance level as an interpretation of the signature  $\Sigma$ .

Example: Signature

- Classes are represented by unary predicates: Emp/1, Mgr/1, AMgr/1, TMgr/1, Dept/1.
- Attributes are represented by binary predicates: name/2, salary/2 (optionally, this could be modeled by unary functions)
- (binary) relationships are represented by binary relationships: wf/2, mg/2, sub/2.

#### Thus,

```
\Sigma_{company} = \{ \text{Emp/1}, \text{Mgr/1}, \text{AMgr/1}, \text{TMgr/1}, \text{Dept/1}, \text{name/2}, \text{salary/2}, \text{wf/2}, \text{mg/2}, \text{sub/2} \}.
```

#### Example: Subclass Constraints

 $\begin{aligned} \forall x : \mathsf{Mgr}(x) &\to \mathsf{Emp}(x) ,\\ \forall x : \mathsf{AMgr}(x) &\to \mathsf{Mgr}(x) ,\\ \forall x : \mathsf{TMgr}(x) &\to \mathsf{Mgr}(x) ,\\ \forall x : \mathsf{Mgr}(x) &\to (\mathsf{AMgr}(x) \lor \mathsf{TMgr}(x)) \text{ since declared as covering} \end{aligned}$ 

#### 510

 $\begin{aligned} & \forall x : (\exists n : name(x, n) \rightarrow (\mathsf{Emp}(x) \lor \mathsf{Dept}(x))) , \\ & \forall x : (\exists s : salary(x, s) \rightarrow (\mathsf{Emp}(x))) , \\ & \forall x, y : (sub(x, y) \rightarrow (\mathsf{Emp}(x) \land \mathsf{Mgr}(y))) , \\ & \forall x, y : (sub(x, y) \rightarrow (\mathsf{Emp}(x) \land \mathsf{Mgr}(y))) , \\ & \forall x, d : (wf(x, d) \rightarrow (\mathsf{Emp}(x) \land \mathsf{Dept}(d))) , \\ & \forall x, d : (mg(y, d) \rightarrow (\mathsf{Mgr}(y) \land \mathsf{Dept}(d))) . \end{aligned}$ 

#### Example: Further Constraints

• a person is subordinate to the manager of each department he/she works for:

 $\forall x, y, d: \mathsf{wf}(x, d) \land \mathsf{mg}(y, d) \land x \neq y \to \mathsf{sub}(x, y)$ 

- should we have mg  $\subseteq$  wf, or mg  $\cap$  wf =  $\emptyset$ ? The first is OK:  $\forall y, d : mg(y, d) \rightarrow wf(y, d)$
- add axioms that guarantee irreflexivity and transitivity for "subordinate":  $\forall x : \neg \operatorname{sub}(x, x)$  and  $\forall x, y, z : (\operatorname{sub}(x, y) \land \operatorname{sub}(y, z) \rightarrow \operatorname{sub}(x, z)).$

Constraints that are not added

• cardinality of "subordinate"? "Every employee has a boss"

 $\forall x: \mathsf{Emp}(x) \to \exists y: \mathsf{sub}(x, y)$ 

This causes a semantical problem with the boss: an infinite chain is needed - leading either to only infinite models, or a cycle.

- add an axiom that guarantees that the company has at least one employee:  $\exists x : \mathsf{Emp}(x)$  – then the set of axioms is unsatisfiable.
- such investigations help to validate an ontology. Ontology design tools allow to check for inconsistency, empty classes etc.

Axiomatization of the "company" scenario

Denote the conjunction of the above formulas by  $Axioms_{Company}$ .

- For any database/knowledge base S using this scenario,  $S \models Axioms_{Company}$  is required.
- a database then describes the individuals and their individual properties in this world.

#### Example: Instances

• The signature is extended by constant symbols for all *named* elements of the domain:

 $\Sigma_{my\_company} := \Sigma_{company} \cup \{\text{Alice/f0, Bob/f0, John/f0, Mary/f0, Tom/f0, ..., Sales/f0, ...}\}$ (Note: the signature symbols are capitalized, wereas alice, bob etc denote the elements

of the domain).

- first-order structure  $\mathcal{S} = (I, \mathcal{D})$  as ...
- Domain  $\mathcal{D} = \{a | ice, bob, john, mary, tom, larry, sales, prod, mgm\}$
- map constant symbols (nullary function symbols) to  $\mathcal{D}$ :  $I(Alice) = alice, I(Bob) = bob, \ldots, I(Sales) = sales, \ldots$
- map unary predicates to subsets of the domain D:  $I(Emp) = \{alice, bob, john, mary, tom, larry\}, I(Mgr) = ..., I(Dept) = ..., ...,$
- map binary predicates to subsets of  $\mathcal{D} \times \mathcal{D}$ :  $I(wf) = \{(alice, sales), (mary, sales), (larry, sales), (bob, prod), (bob, sales), (tom, prod), (john, mgm) \}$ I(mg), I(sub), I(name), I(salary) see Slide 505.

514

## **AXIOMATIZATION OF A SCENARIO**

The axiomatization of a scenario (general formulas + a given instance) consists of

- the general axiomatization of the entity types/classes and relationships,
- · literals describing the individuals (class membership and relationships),
- · closure formulas that state that no other things/relationships exist.

#### DB Queries vs. Reasoning

• queries:

- for which X, Y does  $\mathcal{S} \models F(X, Y)$  hold?
- against a single instance,
- this allows algebraic evaluation (if the query can expressed in the algebra).
- reasoning:
  - for which X, Y does a specification  $\mathcal{F}$  logically imply that F(X, Y) holds?
  - There are all models of  $\mathcal{F}$  considered (although, if it is a complete specification with closure axioms, there is only one model).
  - allows "querying" general specifications for e.g. guaranteeing correctness properties.

#### Example: Company Ontology

The axiomatization of "my company" with the given individuals is then

 $\begin{array}{ll} \mathsf{Axioms}_{\mathsf{C}ompany} \land & \mathsf{Emp}(\mathsf{Alice}) \land \mathsf{Emp}(\mathsf{Bob}) \land \ldots \land \mathsf{Dept}(\mathsf{Sales}) \land \ldots \land \\ & \mathsf{wf}(\mathsf{Alice},\mathsf{Sales}) \land \ldots \land \mathsf{mg}(\mathsf{Alice},\mathsf{Sales}) \land \ldots \land \\ & \forall x : \mathsf{Emp}(x) \rightarrow (x = \mathsf{Alice} \lor x = \mathsf{Bob} \lor \ldots) \land \\ & \forall x : \mathsf{Dept}(x) \rightarrow (x = \mathsf{Sales} \lor x = \mathsf{Prod} \lor \ldots) \land \\ & \forall x, y : \mathsf{wf}(x, y) \rightarrow ((x = \mathsf{Alice} \land y = \mathsf{Sales}) \lor \ldots) \land \ldots \end{array}$ 

#### Example: Instances (Alternative)

 alternatively, instead of constant symbols, all individuals can be described by *existentially* quantified variables:

#### $Axioms_{Company} \land$

```
\exists alice, bob, \dots, sales, \dots: (name(alice, "Alice") \land \dots \land Emp(alice) \land Emp(bob) \land \dots \land Dept(sales) \land \dots \land Mgr(alice) \land \dots \land wf(alice, sales) \land \dots \land mg(alice, sales) \land \dots \land sub(mary, alice) \land \dots)
```

#### 516

# 9.2 Logical Entailment

#### **Definition 9.1**

Let *F* and *G* two (closed) formulas over a signature  $\Sigma$ . We write

 $F \models G$  (F logically entails G)

if for each structure S over  $\Sigma$ , if  $S \models F$  then also  $S \models G$ .

Example

 $\forall x : ((p(x) \to q(x)) \land (q(x) \to r(x)))) \models \forall x : (p(x) \to r(x))$ 

Logical Entailment as Proof

- usually F is a "large" conjunctive formula, containing the specification, and G is a "claim" to be shown to be a logical consequence of F.
- note: for an interpretation  $\mathcal{I}$ , the notation  $\mathcal{I} \models F$  from the previous section can be regarded as representing  $\mathcal{I}$  by its axiomatization  $\phi_{\mathcal{I}}$  (cf. Slide 515) and then

$$\mathcal{I} \models F \iff \phi_{\mathcal{I}} \models F .$$

## LOGICAL ENTAILMENT IN A KNOWLEDGE BASE

- for a FOL knowledge base, it is not always necessary to give all facts explicitly,
- axioms and *some* "basic" facts are often sufficient,
- further facts can be proven/added to the KB by logical entailment,
- further universally quantified formulas can be derived,
- entailment is also relevant when verifying consistency (satisfiability) of an ontology specification.

(most of this: see later)

How to Prove Entailment?

- it is not necessary (and not possible) to compute all models of F to check if  $F \models G$ , instead
- prove it on the semantical level by symbolic reasoning ...

518

## LOGICAL ENTAILMENT: EXAMPLE

Consider Axioms\_ $Company \land mg(Alice, Sales)$ .

Does Emp(Alice) hold in each model S = (D, I) of Axioms<sub>Company</sub> (= is it logically entailed by the axioms)?

- $S \models mg(Alice, Sales)$  implies  $(I(Alice), I(Sales)) \in I(mg)$ , i.e.,  $(alice, sales) \in I(mg)$ .
- S ⊨ ∀y, d : mg(y, d) → wf(y, d) (axiom) implies that for all d<sub>1</sub>, d<sub>2</sub> ∈ D, S ⊨<sub>{y/d1,d/d2}</sub> mg(y, d) → wf(y, d) which means that if S ⊨<sub>{y/d1,d/d2}</sub> mg(y, d), then also S ⊨<sub>{y/d1,d/d2}</sub> wf(y, d). The former is equivalent to (d<sub>1</sub>, d<sub>2</sub>) ∈ I(mg) that we have shown above for (alice, sales). Thus, we know that (alice, sales) ∈ I(wf).
- With the same argument as above, use the axiom
   S ⊨ ∀x, d : (wf(x, d) → (Emp(x) ∧ Dept(d))) for concluding that alice ∈ I(Emp) which means that S ⊨ Emp(Alice).

## REASONING

• *prove* by symbolic reasoning that a formula is *implied* by a knowledge base: Algorithms for deriving entailed facts or formulas, or for checking entailment by automated, symbolic reasoning.

## VALIDITY AND DECIDABILITY

- · preferably use a decidable logic/formalism
- with a complete calculus/reasoning mechanism
- · Propositional logic: decidable
- · First-order logic: undecidable
- Horn subset (= positive rules, with a special model theory) of FOL: decidable; with negation in the body: still decidable
- · 2-variable-subset of FOL: decidable
- Description Logic subsets of FOL: range from decidable to undecidable

520

## A SIMPLE NATURAL REASONING SYSTEM

Inference rule "Modus Ponens":

$$rac{orall ar x:(P o Q) \ , \ P'}{\sigma(Q)}$$
 where  $P' = \sigma(P)$  for a substitution  $\sigma$ .

Consider again the derivation from Slide 519:

 $\frac{\forall y, d : (\mathsf{mg}(y, d) \to \mathsf{wf}(y, d)) , \ \mathsf{mg}(\mathsf{Alice}, \mathsf{Sales})}{\mathsf{wf}(\mathsf{Alice}, \mathsf{Sales})}$  $\forall x, d : (\mathsf{wf}(x, d) \to (\mathsf{Emp}(x) \land \mathsf{Dept}(d))) , \ \mathsf{wf}(\mathsf{Alice}, \mathsf{Sales})$  $\mathsf{Emp}(\mathsf{Alice}) \land \mathsf{Dept}(\mathsf{Sales})$ 

- forward-reasoning,
- uses only knowledge by implication rules "if ... then ...".
  - no disjunction (disjunction is not part of daily common reasoning, but merely part of puzzles like Sudoku),
  - no existential quantification (reasoning with things that are known to exist but cannot be named explicitly)

towards automated symbolic reasoning:					
Prenex Form					
<b>Definition 9.2</b> A formula <i>F</i> is in prenex form if it has the form					
$F = Q_1 x_1 \ Q_2 x_2 \ \dots \ Q_n x_n : G$					
where each $Q_i \in \{\forall, \exists\}$ is a quantifier and the $x_i$ are variables, and G is quantifier-free.					
<b>Theorem 9.1</b> For each formula $F$ , there is an equivalent formula $F'$ which is in prenex form.					
Proof: by induction. Pull quantifiers to the outside.					
Next step: handling existentials:					
• Consider $\forall x : (Emp(x) \to \exists d : wf(x, d)) \text{ and } \forall d : (Dept(d) \to \exists m : mg(m, d))$					
<ul> <li>for x = Alice: "the department Alice works for", and "the person that manages this department"</li> </ul>					
$\Rightarrow$ implicit way to name things: $f_{dept}(Alice)$ and $f_{manages}(f_{dept}(Alice))$ .					
522					
SKOLEMIZATION					

Consider formula F from Example 8.6(3):  $F = \forall x \exists y : p(x, y)$ .

When talking about models of it, "given a certain  $x_1$ , there is an  $y_1$  such that ...".

One can imagine a (new) function f which returns "the (or one of them, if there are many) y for a given x":  $f(x_1) := y_1$ .

## Definition 9.3 (Skolem Form)

[after Thoralf Skolem, a Norwegian Mathematician]

For a formula F in prenex form, its Skolem form sk(F) is defined as follows:

For each subformula of the form  $G = \exists y : H(x_1, \dots, x_n, y)$  where the  $x_i$  are the free variables of H that are universally quantified by a subformula F' of F that contains G, replace each occurrence of y by the term  $f(x_1, \dots, x_n)$  where f is a new function symbol.

## Example 9.1

Consider  $F = \forall x \exists y \forall z : p(x, y, z)$ . For skolemizing y consider  $G(x) = \exists y \forall z : p(x, y, z)$  and replace y by f(x).  $sk(\forall x \exists y \forall z : p(x, y, z)) = \forall x, z : p(x, f(x), z)$ 

#### Notes

- The definition is originally applied only to prenex normal form (i.e. all quantifiers on top, quantifier-free body);
- it holds in the same way for formulas that are not in prenex form (but the proof uses prenex form).
- There is an improvement: take only those universally quantified variables that are free in the body of the respective quantified subformula  $\exists y : F(x_{i_1}, \ldots, x_{i_m})$ .
- Further examples: Slide 528.

#### Usage of Skolemization

The formula sk(F) is obviously not equivalent with F (it even uses an extended signature), but:

#### Theorem 9.2

For every formula F (in prenex form), sk(F) is satisfiable if and only F is satisfiable.

Idea: extend the interpretation I with the new function symbols by mapping  $f(d_1, \ldots, d_n)$  to a  $d \in \mathcal{D}$  which exists for the given  $x_1, \ldots, x_n$ .

524

#### Proof ... needs a definition and a lemma before:

## **Definition 9.4 (Substitution)**

A substitution is a mapping  $\sigma$ : Variables  $\rightarrow$  Term<sub> $\Sigma$ </sub>.

For a formula *F*, variables  $x_1, \ldots, x_n$  and terms  $t_1, \ldots, t_n$ , the application of  $\sigma = [x_1/t_1, \ldots, x_n/t_n]$  to *F*, written as  $F[x_1/t_1, \ldots, x_n/t_n]$ , replaces every free occurrence of  $x_i$  in *F* by  $t_i$ .

A substitution is collision-free if the mapped variables do not occur in any of the replacement terms.

Example:  $\sigma = [x/f(a), y/g(v, 3), z/f(g(a, w))]$  is collision-free, and  $\sigma(p(x, y, z)) = p(f(a), g(v, 3), f(g(a, w))).$ 

## Lemma 9.1 (Substitution Lemma)

For every structure S, and every variable assignment  $\beta$ , every terms s, t, every variable x, and every formula F,

525

- $\mathcal{S}(s[x/t],\beta) = \mathcal{I}(s,\beta^d_x)$  where  $d = \mathcal{S}(t,\beta)$ ,
- $\mathcal{S} \models_{\beta} F[x/t] \Leftrightarrow \mathcal{S} \models_{\beta_x^d} F$  where  $d = \mathcal{S}(t,\beta)$ .

(Proof by structural induction over s and F)

#### Proof of Theorem 9.2

Induction over the number of replacements, top-down (replacing outer  $\exists$ -quantifiers first). Consider *F* in prenex form

 $F = \forall x_1, \dots, x_n \exists y : G(x_1, \dots, x_n, y)$ , and its skolemization  $F' = \forall x_1, \dots, x_n \exists y : G[y/f(x_1, \dots, x_n)].$ 

" $\Rightarrow$ ": Assume that *F* is satisfiable; there exists a structure S = (I, D) s.t.  $S \models F$ .

Thus, for all  $d_1, \ldots, d_n \in \mathcal{D}$ ,

$$\mathcal{S} \models_{\beta} \exists y : G(x_1, \dots, x_n, y)$$

where  $\beta = \{x_1 \mapsto d_1, \dots, x_n \mapsto d_n\}.$ 

This is exactly the case if there is a  $d \in D$  (dependent on the  $d_i$ ) s.t.

$$\mathcal{S}\models_{\beta_y^d} G(x_1,\ldots,x_n,y)$$
.

Define a new structure S' = (I', D) such that I' coincides with I wherever I is defined, and defines the new *n*-ary function symbol f to map every *n*-tuple  $(d_1, \ldots, d_n)$  to the above d (which depends on the  $d_i$ ).

(using the Axiom of Choice which guarantees the existence of this function).

#### 526

#### Proof of Theorem 9.2

" $\Rightarrow$ " (cont'd): By the substitution lemma,

 $\mathcal{S}' \models_{\beta} G(x_1, \dots, x_n, y)[y/f(x_1, \dots, x_n)]$ 

an since this holds for all  $d_1, \ldots, d_n \in \mathcal{D}$ ,

 $\mathcal{S}' \models \forall x_1, \dots, \forall x_n : G(x_1, \dots, x_n, y)[y/f(x_1, \dots, x_n)],$ 

i.e.,  $S' \models F'$  – and F' is satisfiable (and there is a constructive description how (a possible) S' is obtained by extending S).

" $\Leftarrow$ ": analogously.

 $\mathcal{S}'(f(x_1,\ldots,x_n))$  is the "witness" element of the domain that satisfies the  $\exists y$ .

What is this good for??

• symbolic reasoning: the skolemized formula is sufficient to develop a "typical" model of the formula with the relationships between its constants in *tableau proofs*.

#### Example 9.2

Consider the formula

$$F = \forall x \exists y : (p(x, y) \land \exists z : (r(z) \land \neg q(x, y, z)))$$

- prenex form:  $\forall x \exists y \exists z : (p(x, y) \land r(z) \land \neg q(x, y, z))$ (aside: this is the same logical transformation as the "push-into-exists" rule of RANF)
- skolemize with  $f_y$  and  $f_z$ :  $sk(F) = \forall x : (p(x, f_y(x)) \land r(f_z(x)) \land \neg q(x, f_y(x), f_z(x))).$
- understand that it is sufficient to describe *z* as *f*(*x*), not as a function of *x* and *y*:
   with *y* → *f*<sub>1</sub>(*x*) and *z* → *f*<sub>2</sub>(*x*, *y*), follows *z* → *f*<sub>2</sub>(*x*, *f*<sub>1</sub>(*x*)) which is effectively a function only of *x*.

#### Example 9.3

Consider the formula

 $F = (\forall x_1 \exists y : p(x_1, y)) \land (\forall x_2 \exists z : (q(x_2, z)))$ 

- prenex form (one alternative):  $\forall x_1, x_2 \exists y, z : (p(x_1, y) \land q(x_2, z))$
- skolemize with  $f_y$  and  $f_z$ :  $\forall x_1, x_2 : (p(x_1, f_y(x_1, x_2)) \land q(x_2, f_z(x_1, x_2)))$ which is a lot longer than necessary!
- skolemize without intermediate prenex step:  $(\forall x_1 : p(x_1, f_y(x_1))) \land (\forall x_2 : q(x_2, f_z(x_2)))$

#### Aside: Skolem Normal Form

## **Definition 9.5 (Skolem Normal Form)**

A formula is in Skolem Normal Form if it is

- closed (i.e., no free variables),
- of the form  $\forall x_1 \dots \forall x_n : B$ ,
- and *B* is in conjunctive normal form (CNF), *i.e.*, of the form  $(a_{11} \lor \ldots \lor a_{1k_1}) \land \ldots \land (a_{m1} \lor \ldots \lor a_{mk_m})$
- every formula F can be transformed in an equivalent formula F' in Skolem Normal Form.

- Skolem Normal Form is used for *Resolution Proofs*.
- in this lecture, we will not apply resolution to arbitrary inputs, but only to *logical rules* (Datalog rules) which come automatically in CNF (and without function symbols).
- the idea of skolemization "on demand" is also used in Tableau proofs.

## REASONING

• *prove* by symbolic reasoning if a formula F is *implied* by a knowledge base  $\mathcal{K}$  (which is a set of closed formulas):

$$\mathcal{K} \models F$$
 ?

• Equivalently, let K denote the conjunction of all formulas in  $\mathcal{K}$ :

 $\emptyset \models K \to F$  does  $K \to F$  hold in all interpretations?

• Equivalently, show that there is no interpretation where  $\neg(K \rightarrow F)$  holds.

#### · Equivalently,

show that  $\neg(K \to F)$  (which is the same as  $K \land \neg F$ , i.e. a model of  $K \land \neg F$  would be a model of K and not of F) is unsatisfiable.

⇒ try to systematically develop a model of  $K \land \neg F$ . If this fails, then,  $K \land \neg F$  is unsatisfiable.

# 9.3 First Order Tableau Calculus

- Systematic construction of a model of a formula.
- Goal: show that this is not possible. Otherwise a counterexample is generated.
- counterexamples can be interpreted as answers to a query.

Start the tableau with a set  $\mathcal{F}$  of formulas:

input set ${\cal F}$						
F	for all $F \in \mathcal{F}$					

The tableau is then extended by expansion rules.

TABLEAU RULES							
Original Definition: Raymond Smullyan: First-Order Logic. Springer, New York, 1968.							
lpha-rule (conjunctive)	: $F \wedge G$	$\neg(F \lor G)$	Clo	osure Rule:			
	$\overline{F}$	$\neg F$		$\sigma(A)$			
	G	$\neg G$		$\neg \sigma(A)$			
eta-rule (disjunctive): $\gamma$ -rule (universal):	$\frac{F \lor G}{F \mid G}$ $\forall x: F$	$ \frac{\neg (F \land G)}{\neg F \mid \neg G} \\ \neg \exists x : F $	_				
where X is a new	F[X/x] v variable.	$\neg F[X/x]$		the variable substitutions in $\sigma$ , see examples below)			
$\delta$ -rule (existential):	$\exists x:F$		$\neg \forall x: F$				
	F[f(free(F))]	$)/x]$ $\neg F$	[f(free(F)	)/x]			
where $f$ is a new Note: $\delta$ -Rule acco Free Variable Set	Skolem functi ording to Rein	ion symbol. er Hähnle, Pet	er H. Schr	nitt: The Liberalized delta-Rule in			

## RESULT

## **Definition 9.6**

A branch T in a tableau T is closed, if it contains the formula  $\bot$ . A tableau T is closed if every branch is closed.

## CORRECTNESS

#### **Definition 9.7**

A Tableau  $\mathcal{T}$  is satisfiable it there exists a structure  $\mathcal{S} = (I, \mathcal{D})$  such that for every assignment  $\beta$  of the free variables there is a branch T in  $\mathcal{T}$  such that  $\mathcal{S} \models_{\beta} T$  holds.

#### Theorem 9.3

If a tableau T is satisfiable, and T' is obtained from T by application of one of the above rules, then T' is also satisfiable.

Examples, Proof: to do in the lecture, sketch of two cases on Slide 535.

Issues: completeness of the method (only possible for decidable logics) and termination of the algorithm: how to detect when a tableau cannot be closed, and to restrict the expansion to promising rule applications.

534

## CORRECTNESS OF THE FOL TABLEAU CALCULUS: PROOF SKETCH

Assume T satisfiable; T' obtained from applying a tableau rule. We show only two cases:

- Disjunction: Application of the rule to a formula of the form A ∨ B. There is an interpretation M such that for each assignments β of free variables, there is some branch T (= the set of formulas on this branch) such that M ⊨<sub>β</sub> T. If T is not the branch of T that is extended in this step, T does not change. Otherwise, M ⊨<sub>β</sub> A ∨ B. By definition, M ⊨<sub>β</sub> A or M ⊨<sub>β</sub> B. Thus, for (at least one) one of the two branches, T<sub>1</sub><sup>\*</sup> or T<sub>2</sub><sup>\*</sup> obtained from the application, M ⊨<sub>β</sub> T<sup>\*</sup>.
- Existential: Application of the rule to a formula of the form  $\exists y : F(X_1, \dots, X_n, y)$  to a branch *T*. Again, consider any  $\beta$  (which assigns  $\beta(X_1), \dots, \beta(X_n)$  to the free variables in *F*) such that  $\mathcal{M} \models_{\beta} T$ .

This means, for every  $\beta(X_1), \ldots, \beta(X_n)$ , there is some element of the universe that "fits" for the existential formula. Extend the signature with a new *n*-ary "Skolem" function  $f_F$  that takes the values of  $X_1, \ldots, X_n$  as input and is interpreted to return the appropriate element (and that returns an arbitrary value for those  $\beta'$  where  $\mathcal{M} \not\models_{\beta'} T$ ).

The extended branch  $T^*$  appends  $F(X_1, \ldots, X_n, f_F(X_1, \ldots, X_n))$  to T.

For the extended interpretation  $\mathcal{M}'$  (which is the same as  $\mathcal{M}$  except for the new function),  $\mathcal{M}' \models_{\beta} T^*$  whenever  $\mathcal{M} \models_{\beta} T$ .

## TABLEAU CALCULUS: EXAMPLE

Consider again the derivation from Slide 519: Does

Axioms<sub>Company</sub>  $\land$  mg(Alice, Sales). imply Emp(Alice)?

- start the tableau with Axioms\_{Company}, mg(Alice, Sales), and the negated claim  $\neg Emp(Alice)$ ,
- see tableau next slide.
- this example: follow human reasoning:
  - the proof steps are known,
  - "apply"  $\forall y, d : (mg(y, d) \rightarrow wf(y, d))$  with y/Alice, d/Sales, "obtain" wf(Alice, Sales)
  - "apply"  $\forall x, d : (wf(x, d) \rightarrow (Emp(x) \land Dept(d)))$  for x/Alice, d/Sales and obtain Emp(Alice).
- the tableau illustrates the application of "rules":
  - close left branch "not body" immediately, propagate closure substitution to the right branch.
- $\Rightarrow$  illustrative, but naive example driven by human *forward reasoning*.

Tableau Calculus: Example
$$\forall y, d: (mg(y, d) \rightarrow wf(y, d))$$
 $\forall x, d: (wf(x, d) \rightarrow (Emp(x) \land Dept(d)))$  $mg(Alice, Sales)$  $\neg Emp(Alice) (**)$  $|$  $mg(Y_1, D_1) \rightarrow wf(Y_1, D_1)$  $/$  $\land$  $\neg mg(Y_1, D_1)$  $wf(Alice, Sales)$  $|$  $wf(X_2, D_2) \rightarrow (Emp(X_2) \land Dept(D_2)))$  $/$  $\neg wf(X_2, D_2)$  $\Box \{X_2 \rightarrow Alice, D_2 \rightarrow Sales\}$  $Emp(Alice) \land Dept(Sales)$  $Emp(Alice) (**)$  $Dept(Sales)$  $\Box (**)$ 

#### **Automated Reasoning**

- · do not close branches immediately by replacing variables
  - if there are multiple possible closing substitutions, keep the variable until the whole tableau can be closed,
- this also illustrates the use of skolem functions better.

#### **EXAMPLE: TABLEAU EXPANSION FOR AN EXISTENTIAL VARIABLE**

Consider again the Company scenario. Show: for every employee x, there is an employee y (x = y allowed) such that sub(x, y) holds. (sketch: for every employee x there is a at least a "primary" department  $f_{dept}(x)$  where this person works, and every department d has a manager  $f_{mg}(d)$  that manages the department and that thus is a subordinate of x).

Note that in case that x works in several departments, any of them can be chosen for  $f_{dept}(x)$ . *e* is subordinate to  $f_{mg}(f_{dept}(x))$ .

Tableau: next slide.

- again: followed human reasoning steps.
- · automated reasoning: how to choose which axioms to use?



## TABLEAU CALCULUS: EXAMPLE

Consider again the tableau proof from Slide 536:

Does Axioms<sub>Company</sub>  $\land$  mg(Alice, Sales) imply Emp(Alice)?

- again, start the tableau with Axioms<sub>Company</sub>, mg(Alice, Sales), and the negated claim  $\neg \text{Emp}(\text{Alice})$ .
- · automated reasoning: how to choose which axioms to use?
- consider the "goal" ¬Emp(Alice):
   clearly, some formula (here: rule) that derives Emp(Alice) is needed. Start with expanding ∀x, d : (wf(x, d) → (Emp(x) ∧ Dept(d))).
- close the right branch with *x*/Alice, get a new "goal" ¬wf(Alice, *D*) in the left branch. Now, some formula (here: rule) that derives wf(Alice, \_) is needed. Expand
   ∀y, d : (mg(y, d) → wf(y, d)).
- see tableau next slide.

Tableau Calculus: Example		
	Axioms <sub>Con</sub>	mpany
	mg(Alice,	Sales)
	¬Emp(Alice	e) (**)
	$\forall x, d : (wf(x, d) \to (Er$	$mp(x) \land Dept(d)))$
	$wf(X_1, D_1) \to (Emp(X_1, D_1))$	$X_1) \wedge Dept(D_1)))$
	/	\
	$\neg wf(X_1, D_1)$	$Emp(X_1) \wedge Dept(D_1)$
	I	$Emp(X_1)$ (**)
	$\neg$ wf(Alice, $D_1$ )	$Dept(D_1)$
	$\forall y, d: (mg(y, d) \to wf(y, d))$	$\Box\{X_1/Alice\}(**)$
	$mg(Y_2, D_2) \to wf(Y_2, D_2)$	
	/	
$\neg m$	$g(Y_2, D_2) \qquad \qquad wf(Y_2, D_2)$	
	$  \qquad \Box \{Y_2 \to Alice, D_2 \to$	$D_1$ }
⊸mg	$(Alice, D_1)$	
$\Box\{D_1$	$1 \rightarrow Sales$	
	-	

#### Comments

Consider again the tableaux from Slides 537 and 541.

- both used only formulas of the form  $P \rightarrow Q$  where P and Q are conjunctions,
  - forward reasoning: close left branch immediately,
  - backward reasoning: close right branch immediately,
  - $\Rightarrow$  linear proofs (if the correct rule is always chosen)
- $\Rightarrow$  preview:

the resolution calculus provides an efficient calculus for such cases where only rules are used (Datalog)

 tableaux have higher expressiveness: can handle full disjunction etc.: Description Logics and OWL (Semantic Web) use tableaux

542

## NON-CLOSED TABLEAUX: (TYPICAL) SAMPLE MODELS

Is the axiom  $\forall x : \neg(p(x) \land q(x))$  together with the "database"  $\{p(a), q(b)\}$  consistent?

$$p(a)$$

$$q(b)$$

$$\forall x : \neg(p(x) \land q(x))$$

$$\neg(p(X_1) \land q(X_1))$$

$$/ \qquad \setminus$$

$$\neg p(X_1) \qquad \neg q(X_1)$$

$$\neg q(x_1) \qquad \neg q(x_1)$$

$$\neg(p(X_2) \land q(X_2))$$

$$/ \qquad \land$$

$$\neg p(X_2) \qquad \neg q(X_2)$$

$$\neg p(b) \qquad \Box \{X_2 \rightarrow b\}$$

- · there is no way to close the tableau
- its non-closed path describes a model of the input formula (where ¬q(a) and ¬p(b) hold which are not specified in the database – open world reasoning)

## TABLEAU CALCULI: APPLICATION FOR QUERY ANSWERING

Consider the database  $\{\forall x : (p(x) \rightarrow q(x)), p(a), q(b)\}$  and the query ? - q(X).

$$orall x : (p(x) \to q(x))$$
 $p(a)$ 
 $q(b)$ 

 $\neg q(X)$  add the negated query with a free variable

- collect all substitutions of *X* that can be used to close the tableau.
- note: the substitution can comprise a the application of a Skolem function. Then, the "answer" can only be described as a thing that satisfies a certain existential formula.

Consider  $\forall x : (person(x) \rightarrow (\exists y : person(y) \land father(x, y))),$  $\forall x, y, z : ((father(x, y) \land father(y, z)) \rightarrow grandfather(x, z)),$ person(john), person(jack), father(john,jack) and the query ?- grandfather(john,X).

544

## TABLEAU CALCULI IN GENERAL

- intuitive idea
- can be designed in this way for any logic (modal logics, description logics etc.)
- · implementations use more efficient heuristics

#### Examples + Exercises

prove that

$$\forall x: ((p(x) \to q(x)) \land (q(x) \to r(x)))) \models \forall x: (p(x) \to r(x))$$

and

 $\forall x: ((p(x) \to q(x)) \land (q(x) \to r(x)))) \models (p(a) \to r(a))$ 

- Consider the italian-vs-english ontology from Slide 546. Consider the statement "all Italians are lazy". Prove it or give a counterexample.
- Consider the italian-professors ontology from Slide 547. Is there anything interesting to prove?







## TABLEAUX AND CONJUNCTIVE QUERY ANSWERING

"All organizations that have their headquarters in the capital of a European member country" Using a simplified Mondial signature:

$$F(org) = \exists cty, ctry(\mathsf{headq}(org, cty, ctry) \land \mathsf{capital}(ctry, cty) \land \mathsf{enc}(ctry, \mathsf{``Europe''}) \land \mathsf{member}(ctry, org))$$

Start the tableau with  $\neg F(X)$ .

Tableaux and  $body \rightarrow head$  Rules

Consider again the tableaux from Slides 537 and 541.

- all used axioms are of the form  $\forall x_1, \ldots, x_n : body \rightarrow head$ ,
- plus a (negated) query ("goal")  $\neg F(\bar{X})$ ,
- standard tableau pattern:

- close one branch immediately (forward: left, backward: right), obtain a set of tuples binding  $(X_1, \ldots, X_n)$  (i.e., that satisfy *body*), propagate to the other branch,
- continue with next axiom.
- Again, the tableau is closed for all bindings of  $\bar{X}$  that are answers.

## SUMMARY: TABLEAU REASONING

- covers full first-order logic,
- theoretically incomplete,
- most practical cases result in acceptable performance,
- · reasons for more complex tableaux:
  - search for proof tableaux/trees
  - disjunction (explore several branches where only one contributes)
  - multiple instantiations of universally quantified variables
    - \* needed for self-joins, transitivity,
    - \* especially in combination with skolemized  $\exists$ -terms.
- simple patterns (rules, conjunctive body/head) result in effectively nearly-algebraic evaluation.
- But: for simple patterns one does not need a full first-order reasoner.

552

## **PROPERTIES OF FIRST-ORDER LOGIC DECISION PROCEDURES**

- calculi (=algorithms) for checking if F ⊨ G (often by proving that F ∧ ¬G is unsatisfiable)
- write  $F \vdash_C G$  if calculus C proves that  $F \models G$ .
- Correctness of a calculus:  $F \vdash_C G \Rightarrow F \models G$
- Completeness of a calculus:  $F \models G \Rightarrow F \vdash_C G$
- there are complete calculi and proof procedures for propositional logic (e.g., Tableau Calculus or Model Checking)
- if a logic is undecidable (like first-order logic) then there cannot be any complete calculus!

#### What to do?

- $\Rightarrow$  use an undecidable logic and a correct, but (theoretically) incomplete calculus.
- e.g. Software Verification.
- $\Rightarrow$  use a decidable logic (i.e., weaker than FOL).
  - often a restricted set of formulas (Description Logic [Semantic Web], Datalog Variants [Database Theory])

## ASIDE: WHY "FIRST-ORDER"-LOGIC?

Recall:

- there is a domain  $\mathcal{D}$ . Functions and precidates talk *about* elements of  $\mathcal{D}$ .
- there is no way to talk *about* functions or predicates.

#### Higher-Order-Logics

- the elements of the domain  $\ensuremath{\mathcal{D}}$  are "first-order things"
- sets, functions and predicates are "second-order things"
- · predicates about predicates are higher-order things
- higher-order logics can be used for reasoning about metadata

## Example

• Transitivity as a property of predicates is second order:  $\forall p : \text{transitive}(p) \rightarrow (\forall x, z : (\exists y : (p(x, y) \land p(y, z)) \rightarrow p(x, z)))$ Note that transitivity of *a certain* predicate is first-order:  $\forall x, z : ((\exists y : (\text{ancestor}(x, y) \land \text{ancestor}(y, z))) \rightarrow \text{ancestor}(x, z))$ 

554

## Aside: Induction Axiom as Example for Second Order Logic

- a well-founded domain d (i.e., a finite set of minimal elements (for which min(d,x) holds) from which the domain can be enumerated by a successor predicate (Natural numbers: 1, succ(i,i+1))
- well-founded: unary 2nd-order predicate over sets
- The induction axiom as a 2nd order logic formula:

$$\begin{split} \forall p, d: \quad (\mathsf{well-founded}(d) \land (\forall x: \min(d, x) \to p(x)) \land (\forall x, y: p(x) \land \mathsf{succ}(x, y) \to p(y))) \to \\ (\forall x: d(x) \to p(x)) \end{split}$$

For natural numbers:

 $\forall p: (p(1) \land (\forall x: p(x) \rightarrow p(x+1))) \rightarrow (\forall x \in \mathbb{N}: p(x))$ 

Aside: Paradoxes can be formulated in 2nd Order Logic

"X is the set of all sets that do not contain themselves"

 $X = \{z : z \notin z\}$ 

A set "is" a unary predicate: X(z) holds if z is an element of X (for example, classes, i.e., Person(x), City(x))

Logical characterization of X:  $X(z) \leftrightarrow \neg z(z)$ ,

applied to z := X - is X in X?  $X(X) \leftrightarrow \neg X(X)$ .

... can neither be true nor false.

How to avoid paradoxes

Paradoxes can be avoided if each variable *either* ranges over first-order things (elements of the domain) or over second-order things (predicates).

556

... now, back into the database area:

# Chapter 10 Datalog Knowledge Bases I

In this section:

- Nonrecursive Datalog with Negation: equivalent to the relational algebra, to the relational calculus and to SQL.
- Stratified Recursive Datalog with Negation equivalent to the relational algebra or SQL with recursion (e.g., transitive closure)

In later sections:

• the really new things: well-founded and stable model semantics.

## **CONJUNCTIVE QUERIES**

- *F*(*X*<sub>1</sub>,...,*X<sub>n</sub>*) = ∃*Y*<sub>1</sub>,...,*Y<sub>m</sub>* : *p*<sub>1</sub>(...) ∧ ... ∧ *p<sub>k</sub>*(...) (note constants and variables may occur in the *p<sub>i</sub>* arguments)
- Note: most systems allow also atomic comparisons over built-in datatypes:  $F(X_1, ..., X_n) = \exists Y_1, ..., Y_m : p_1(...) \land ... \land p_k(...) \land atomic comparisons$
- equivalent: SPJR-Algebra (selection, projection, join, renaming)
- SQL: broad SELECT  $X_1, \ldots, X_n$  FROM  $p_1, \ldots, p_n$  WHERE *cond* where *cond* contains the join conditions and selection conditions
- efficient evaluation using indexes etc.
- Restricted expressiveness:
  - only very restricted negation (if at all) of the form  $x_i \theta x_j$ ,  $x_i \theta c$  where  $\theta \in \{\neq, <, \leq, >, \geq\}$
  - no negation/set difference,
  - no universal quantification,
  - no disjunction/set union,
  - no recursion/no transitive closure.

558

## **XSB:** LET'S START WITH CONJUNCTIVE QUERIES

- a PROLOG dialect developed at State Univ. of NY at Stony Brook (SUNYSB). (so one can actually do everything that is allowed in PROLOG, but we use only Datalog)
- XSB extends the original SB-PROLOG with tabled resolution and HiLog (higher-order logic programming).
- **Open source**: http://xsb.sourceforge.net/

#### Starting XSB at IFI

Installed in the CIP Pool:

- alias xsb='rlwrap ~dbis/LP-Tools/XSB/bin/xsb'  $\leftarrow$  put this into .bashrc
- user@bla:~\$ xsb

```
[xsb_configuration loaded]
[sysinitrc loaded]
XSB Version 3.3.4 (Pignoletto) of July 2, 2011
[i686-pc-linux-gnu 32 bits; mode: optimal; engine: slg-wam; scheduling: local]
[Patch date: 2011/07/08 04:32:08]
| ?-
```

- ?- [mondial]. loads the content of a file (from the current directory).
- ?- country(CN,C,Pop,Area,Cap,CapProv). state a query
- <return>to return to XSB shell
- any key + <return>to get next answer
- CTRL-D: leave XSB

560

#### **Datalog Syntax**

Consider a CQ with only atoms in the body (i.e., positive!)

$$F(X_1,\ldots,X_n) = \exists Y_1,\ldots,Y_m : cq(X_1,\ldots,X_n,Y_1,\ldots,Y_m)$$

Write

```
?- cq(X_1, \ldots, X_n, \underline{Y}_1, \ldots, \underline{Y}_m).
```

where

- the  $X_i$  are the free variables,
- replace  $Y_i$  by  $Y_i$  if it occurs in more than one atom,
- replace  $Y_i$  by \_ if it occurs only once ("don't-care-variables").

Example: countries whose population is > 1000000 and the capital population is not known:

?- country(CN,C,\_Cap,\_CapProv,\_,\_Pop), city(\_Cap,C,\_CapProv,null,\_,\_,),
 \_Pop > 1000000.

Note: null is not a built-in XSB term, but just a constant like 1, bla, 'Bla'.

# 10.1 Datalog Positive Conjunctive Queries – Formal Semantics

#### **Definition 10.1**

Given a relational schema  $\mathbf{R}$  and a (safe) "pure" CQ with only relational atoms in the body (i.e., positive!, no comparisons)

$$F(X_1,\ldots,X_n) = \exists Y_1,\ldots,Y_m : r_1(\bar{u}_1) \land \ldots \land r_k(\bar{u}_k) \qquad r_i \in \mathbf{R}$$

whose Datalog syntax is  $q(X_1, \ldots, X_n) := r_1(\bar{v}_1), \ldots, r_k(\bar{v}_k)$ . (note that the  $\bar{v}_j$  contain  $X_i$  and "\_Y<sub>i</sub>"-variables, the "\_" don't-care, and constants), its answer relation wrt. a database state S is

 $\mathcal{S}(q) := \{ (\beta(X_1), \dots, \beta(X_n)) \mid \beta(\bar{u}_i) \in \mathcal{S}(r_i) \text{ for all } 1 \le i \le k \} .$ 

#### 

#### **Proposition 10.1**

- S(q) contains only values from  $ADOM(S \cup q)$ ,
- For positive conjunctive queries, the Datalog semantics coincides with the classical FOL semantics:

$$\mathcal{S}(q) := \{ (\beta(X_1), \dots, \beta(X_n)) \mid \mathcal{S} \models_{\beta} F(X_1, \dots, X_n) \}$$

562

# 10.2 Positive Datalog: Views as Rules

#### Conjunctive queries as View Definitions

A Datalog "knowledge base"  ${\cal K}$  (also called a Datalog program) consists of

- facts of the form:  $r(c_1, \ldots, c_n)$  (SQL equivalent: the tuples in the database),
- rules of the form  $p(X_1, \ldots, X_k) \leftarrow \exists X_{k+1}, \ldots, X_n : Q(X_1, \ldots, X_n)$ where *p* is a *k*-ary predicate and *Q* is a conjunctive (positive!) query.
  - means: "whenever  $Q(X_1, \ldots, X_n)$  holds for some  $X_{k+1}, \ldots, X_n$ , also  $p(X_1, \ldots, X_k)$  is assumed to hold".
  - SQL equivalent: p is a view.

The signature  $\boldsymbol{\Sigma}$  is partitioned into two sets:

- $\Sigma_{EDB}$ : predicates that occur only in the body of rules ("extensional database" – the interpretation of these predicates is given as facts in the knowledge base)
- $\Sigma_{IDB}$ : predicates that occur in the head (and possibly also in the body) of rules ("intensional database" the interpretation of these predicates is derived from the rules)

#### **XSB** Example

- compiler directive: :- include(filename). (note: no "-" in the filename allowed)
- comments: %

```
564
```

## SEMANTICS OF A DATALOG KNOWLEDGE BASE

The formal semantics is given by Herbrand Interpretations (cf. Slide 497):

Herbrand Interpretation

- the domain consists of constant symbols and datatype literals.
- an interpretation H is explicitly seen as a *finite* set of ground atoms over the predicate symbols and the Herbrand Domain:

```
country (ger, ``Germany'', ``D'', berlin, 356910, 83536115), \quad encompasses (ger, eur, 100).
```

```
\mathcal{H} \models \text{encompasses}(\text{ger}, \text{eur}, 100) \text{ if and only if } (\text{ger}, \text{eur}, 100) \in \text{encompasses}
if and only if \text{encompasses}(\text{ger}, \text{eur}, 100) \in \mathcal{H}.
```

## Examples

- {country(ger, "Germany", "D", berlin, 356910,83536115), country(aut, "Austria", "A", vienna, 83850,8023244), ..., border(aut,ger,784), border(aut,hun,366), ...}
- the file mondial.P has the same schema as Mondial for SQL and uses only atomic values with keys/foreign keys.



## 10.2.1 The Fixpoint Approach to Positive Datalog

Consider a *positive* program (i.e., rules without negation).

• facts of the form  $p(a_1, \ldots, a_n)$  can also be seen as rules:

$$p(a_1,\ldots,a_n)$$
 :- true

"if true holds (which is always the case) then also  $p(a_1, \ldots, a_n)$  must hold".

· application of rules:

The set of ground atoms that is derivable by a rule  $H \leftarrow B_1 \land \ldots \land B_k$  wrt. a given Herbrand Interpretation  $\mathcal{H}$  is formally specified as follows:

{  $\sigma(H)$ :  $\sigma$  is a ground substitution and there is a rule

$$H \leftarrow B_1 \land \ldots \land B_k$$
 in P such that  $\sigma(B_1), \ldots, \sigma(B_k) \in \mathcal{H}$  }

## Example

Let  $\ensuremath{\mathcal{H}}$  contain the facts from mondial.P. The rule

orgOnCont(O,Cont) :- isMember(C,O,\_), encompasses(C,Cont,\_).

with  $\sigma = \{C \mapsto "D", O \mapsto "EU", Cont \mapsto "Europe"\}$  where  $isMember("D", "EU", "member") \in \mathcal{H}$ and where  $encompasses("D", "Europe", 100) \in \mathcal{H}$  derives the atom orgOnCont("EU", "Europe").

#### Bottom-Up-Semantics of Positive Datalog Programs

Consider a *positive* program *P* (i.e., facts, and rules without negation).

- (ground (i.e. without variables)) facts of the form  $p(a_1, \ldots, a_n)$ ,
- (non-ground) rules of the form *head* :- *body*.

#### Definition 10.2 ( $T_P$ -Operator)

For a (positive) Datalog program P and a set I of ground atoms,

 $T_P(I) := \{ \sigma(H) : \sigma \text{ is a ground substitution and there is a rule} \}$ 

 $H \leftarrow B_1 \land \ldots \land B_k$  in *P* such that  $\sigma(B_1), \ldots, \sigma(B_k) \in I$ 

 $T_P(I)$  is called the "immediate consequence operator" since it takes I and applies the rules once.  $T_P^0(I) := I$ 

$$T_P^{n(I)} := T_P(I)$$

$$T_P^{n+1}(I) := T_P(T_P^n(I))$$

$$T_P^{\omega}(I) := \bigcup_{n \in \mathbb{N}} T_P^n(I) \quad infinite \text{ union}!$$

- $T_P^\omega := T_P^\omega(\emptyset)$  usually, start with  $\emptyset$   $^{n \in \mathbb{N}}$
- Intuition: The set  $T_P^{\omega}$  contains all ground facts that can be derived from the program.
- note:  $T_P^1(\emptyset)$  contains the ground facts listed in the program.

#### 568

#### T<sub>P</sub>: Some Straightforward Examples

• Consider the program  $P = \{p, q \leftarrow p, r \leftarrow q, s \leftarrow r \land q\}$ :

$$T_P^1(\emptyset) = T_P(\emptyset) = \{\mathbf{p}\},$$

- $T_P^2(\emptyset) = T_P(\{p\}) = \{p, q\}, -note: p \text{ is derived again}$
- $T_P^3(\emptyset) = T_P(\{\mathsf{p},\mathsf{q}\}) = \{\mathsf{p},\mathsf{q},\mathsf{r}\}$
- $T_P^4(\emptyset) = T_P(\{\mathsf{p},\mathsf{q},\mathsf{r}\}) = \{\mathsf{p},\mathsf{q},\mathsf{r},\mathsf{s}\}$
- $T_P^5(\emptyset) = T_P(\{\mathsf{p},\mathsf{q},\mathsf{r},\mathsf{s}\}) = \{\mathsf{p},\mathsf{q},\mathsf{r},\mathsf{s}\}$

• Consider the program  $Q = \{p(1,2), p(2,3), p(3,4), p(3,5), p(1,6), tc(X,Y) \leftarrow p(X,Y), tc(X,Y) \leftarrow tc(X,Z) \land p(Z,Y)\}:$ Let  $EDB := T_P^1(\emptyset) = \{p(1,2), p(2,3), p(3,4), p(3,5), p(1,6)\}$  for the ground facts.  $T_Q^2(\emptyset) = EDB \cup \{tc(1,2), tc(2,3), tc(3,4), tc(3,5), tc(1,6)\},$  $T_Q^3(\emptyset) = EDB \cup \{tc(1,2), tc(2,3), tc(3,4), tc(3,5), tc(1,6), tc(1,3), tc(2,4), tc(2,5)\},$ 

$$T_Q^4(\emptyset) = EDB \cup \{ tc(1,2), tc(2,3), tc(3,4), tc(3,5), tc(1,6), tc(1,3), tc(2,4), tc(2,5), tc(1,4), tc(1,5) \} = T_Q^5(\emptyset)$$

#### T<sub>P</sub>: Non-Straightforward Examples

Obvious: (Positive) programs with no facts will not derive anything when started with  $\emptyset$ .

- Consider the program P = {p ← p}:
  - $T_P^1(\emptyset) = T_P(\emptyset) = \emptyset = T_P^2(\emptyset) = T_P^{\omega}(\emptyset)$
  - when not starting with  $\emptyset$ , but with {p}:  $T_P^1(\{\mathbf{p}\}) = \{\mathbf{p}\} = T_P^2(\{\mathbf{p}\}) = T_P^{\omega}(\{\mathbf{p}\})$
- Consider the program  $P = \{p \leftarrow q, q \leftarrow p, r \leftarrow p \land q\}$ :

- 
$$T_P^1(\emptyset) = \{\emptyset\} = T_P^2(\emptyset) = T_P^\omega(\emptyset)$$
.

- $T_P^1(\{\mathbf{p}, \mathbf{q}\}) = \{\mathbf{p}, \mathbf{q}, \mathbf{r}\} = T_P^2(\{\mathbf{p}, \mathbf{q}\}) = T_P^{\omega}(\{\mathbf{p}, \mathbf{q}\})$ .
- $T_P^1(\{\mathbf{p}\}) = \{\mathbf{q}\},$   $T_P^2(\{\mathbf{p}\}) = T_P(T_P^1(\{\mathbf{q}\})) = \{\mathbf{p}\},$   $T_P^3(\{\mathbf{p}\}) = T_P(T_P^2(\{\mathbf{p}\})) = T_P(\{\mathbf{p}\}) = \{\mathbf{q}\},$   $\dots \text{ the sequence then alternates } \dots$   $T_P^{\omega}(\{\mathbf{p}\}) = \bigcup_{n \in \mathbb{N}} T_P^n(\{\mathbf{p}\}) = \{\mathbf{p}, \mathbf{q}\}, \text{ which is not a model of } P! \quad (\text{r is missing!})$   $T_P^1(\{\mathbf{r}\}) = \{\emptyset\} = T_P^2(\{\mathbf{r}\}) = T_P^{\omega}(\{\mathbf{r}\}) .$
- ⇒ Starting with  $I \neq \emptyset$  might show strange behaviour. Don't do that. The argument is used only for the iteration  $T_P(T_P^n(\emptyset))$ .

Some Theoretical Properties of  $T_P$ 

Proposition 10.2

 $T_P^{\omega} \models P.$ 

Proof:

- for all facts  $R(c_1, \ldots, c_n)$  contained in  $P, T_P^{\omega} \models R(c_1, \ldots, c_n)$ , i.e.,  $R(c_1, \ldots, c_n) \in T_P^{\omega}$  $(R(c_1, \ldots, c_n) \in T_P^1(\emptyset)).$
- for all rules  $p(X_1, \ldots, X_k) \leftarrow \exists X_{k+1}, \ldots, X_n : Q(X_1, \ldots, X_n)$  contained in P,  $T_P^{\omega} \models \forall X_1, \ldots, X_n : p(X_1, \ldots, X_k) \leftarrow \exists X_{k+1}, \ldots, X_n : Q(X_1, \ldots, X_n)$ : If  $T_P^n(\emptyset) \models_{\beta} \exists X_{k+1}, \ldots, X_n : Q(X_1, \ldots, X_n)$ , then  $T_P^{n+1}(\emptyset) \models_{\beta} p(X_1, \ldots, X_k)$  by definition of  $T_P$ .

... so  $T_P^{\omega}$  looks good. Is it special? What about the infinite union?

#### Some Theoretical Properties of $T_P$

## **Proposition 10.3**

 $T_P$  is monotonous (recall, for positive P), i.e., if  $I_1 \subseteq I_2$  then  $T_P(I_1) \subseteq T_P(I_2)$ .

- As a consequence of this,  $T_P^{n+1}(\emptyset) \supseteq T_P^n(\emptyset)$ .
- $\bigcup_{n \in \mathbb{N}} T_P^n(\emptyset) = \lim_{n \to \infty} T_P^n(\emptyset)$  ... (infinite?) iteration does it stop somewhere?
- Let *HB<sub>P</sub>* denote the Herbrand Base of *P*, i.e., the set of all ground instances of predicates in *P* over the Herbrand universe (which consists of all constants occurring in the atoms in *P*).
   Then, *HB<sub>P</sub>* ⊨ *P* for every positive *P*.

•  $T_P^n(\emptyset) \subseteq \mathcal{HB}_P$  for all  $n \in \mathbb{N}$ .

..... a monotonously growing sequence is bounded from above:

## Theorem 10.1

For some (finite)  $n \in \mathbb{N}$ , a fixpoint, i.e.,  $T_P(T_P^n(\emptyset)) = T_P^n(\emptyset)$  is reached after finitely many steps. For this n,  $T_P^{\omega} = T_P^n(\emptyset)$ .

- $T_P^{\omega}$  can effectively be computed ("bottom-up"),
- queries are then stated against  $T_P^{\omega}$ .

#### 572

Some Theoretical Properties of  $T_P$ 

This is the general definition of the term "fixpoint":

## **Definition 10.3**

For an operator  $\Psi$  mapping from any mathematical domain  $\mathcal{X}$  to  $\mathcal{X}$ , a fixpoint is any x such that  $\Psi(x) = x$ .

## Example:

 $\sqrt{.}$  is an operator  $\mathbb{R} \to \mathbb{R}$ .  $\sqrt{1} = 1$  is a fixpoint of it.  $T_P$  is an operator from sets of ground atoms (i.e., Herbrand interpretations) to sets of ground atoms.

#### Some Theoretical Properties of $T_P$

#### **Proposition 10.4**

For a Datalog program P,

- a) every fixpoint  $\mathcal{F}$  of  $T_P$ , i.e.,  $T_P(\mathcal{F}) = \mathcal{F}$ , is a model of P (but not every model is a fixpoint!), and
- b) for every model  $\mathcal{H}$  of a Datalog program P,  $T_P(\mathcal{H}) \subseteq \mathcal{H}$ .

#### Proof:

- a) Since  $\mathcal{F}$  is a fixpoint,  $T_P(\mathcal{F}) \supseteq \mathcal{F}$ , i.e. it contains all facts in P, and all instances of heads of applicable rule instances. Thus, it is a model of P.
- b) by definition of  $T_P$ :  $\mathcal{H}$  is a model of P, so it already contains all ground instances of heads of applicable rule instances.

Note: a model can also contain additional ground atoms (=facts) that are not required ("supported") by the program, as long as it contains also their consequences. It is still a model.

[Example see next slide]

Outlook: the "Minimal Model" will be a distinguished model (later, the "Well-Founded Model" and "Stable Models" continue this idea of minimality).

#### Models of a Program

Further models of a program can be obtained by adding additional facts (they must be complete wrt. consequences from these).

## Example 10.1

Consider  $P = \{q(X) := p(X); r(X) := q(X); p(a)\}.$ 

- Let  $\mathcal{M} := T_P^{\omega}(\emptyset) = \{p(a), q(a), r(a)\}.$
- Other, bigger models are  $\mathcal{M}_1 = \{p(a), q(a), r(a), r(b)\}$  and  $\mathcal{M}_2 = \{p(a), q(a), r(a), q(b), r(b)\}.$
- Note that N = {p(a), q(a), r(a), q(b)} ⊋ M, but it is not a model.
   Since N ⊆ M<sub>2</sub>, it can obviously be extended to a model (cf. Slide 579).
- The  $\mathcal{M}_i$  are not fixpoints of  $T_P$ :  $T_P(\mathcal{M}_1) = \{p(a), q(a), r(a)\} = \mathcal{M} \subsetneq \mathcal{M}_1$ , and  $T_P(\mathcal{M}_2) = \{p(a), q(a), r(a), r(b)\} = \mathcal{M}_1 \subsetneq \mathcal{M}_2$ .
  - in both cases, according to Proposition 10.4,  $T_P(\mathcal{M}_i) \subseteq \mathcal{M}_i$ , shows that they are models, i.e., all rules are satisfied.
  - The "⊊" shows that some fact has been "invented" which is not forced ("supported") by the rules.
- Usually, fixpoints which are non-minimal models occur if the program contains some "self-supporting" rule  $p \leftarrow p$ .
Some Theoretical Properties of  $T_P$ 

# Definition 10.4

For two Herbrand interpretations,  $\mathcal{H}_1$  and  $\mathcal{H}_2$ ,  $\mathcal{H}_1 \leq \mathcal{H}_2$  if  $\mathcal{H}_1 \subseteq \mathcal{H}_2$ .

# **Proposition 10.5**

 $T_P^{\omega}$  is the least fixpoint of  $T_P$ .

## Proof:

By Proposition 10.4, every fixpoint  $\mathcal{F}$  is a model of P. To be a model of P,  $\mathcal{F}$  contains all facts in P, i.e.,  $\mathcal{F} \supseteq T_P(\emptyset)$ . By induction,  $\mathcal{F} \supseteq T_P^n(\emptyset)$  for each  $n \in \mathbb{N}$ . Thus,  $\mathcal{F} \supseteq T_P^{\omega}$ .

(the full PROLOG case, where the  $\mathcal{HB}_P$  argument does not hold and  $T_P^{\omega}$  is not necessarily finite, follows from monotonicity by the Knaster-Tarski Theorem (fixpoint theory over complete lattices).)

Aside:  $T_P^{\omega}$  in PROLOG

- PROLOG allows function symbols.
- Consider the program  $P := \{ p(a), p(f(X)) \leftarrow p(X) \}$ :  $T_P^{\omega} = \{ p(f^n(a)) | n \in \mathbb{N} \}$  is infinite.

#### 576

## Example/Exercise

Consider the following (recursive) program (including atomic facts and rules):

- $$\begin{split} P &= \{ \begin{array}{ll} \mathsf{country}(a). \ \mathsf{country}(b). \ \mathsf{country}(ch). \ \mathsf{country}(d). \ \mathsf{country}(e). \ \mathsf{country}(f). \ \ldots \\ & \mathsf{border}(a,d). \ \mathsf{border}(a,h). \ \mathsf{border}(a,i). \ \mathsf{border}(d,f). \ \mathsf{border}(i,f). \\ & \mathsf{border}(ch,f). \ \mathsf{border}(ch,a). \ \mathsf{border}(ch,d). \ \mathsf{border}(ch,i). \ \mathsf{border}(e,f). \ \mathsf{border}(p,e). \\ & \mathsf{border}(h,ua). \ \mathsf{border}(ua,r). \ \mathsf{border}(ra,br). \ \mathsf{border}(bol,ra). \ \mathsf{border}(bol,br). \\ & \mathsf{border}(Y,X) \leftarrow \mathsf{border}(X,Y). \\ & \mathsf{reachable}(X,Y) \leftarrow \mathsf{border}(X,Y). \\ & \mathsf{reachable}(X,Y) \leftarrow \mathsf{reachable}(X,Z), \mathsf{border}(Z,Y). \end{array} \end{split}$$
- Give  $T_P^0(\emptyset), T_P^1(\emptyset), T_P^2(\emptyset), \dots, T_P^{\omega}(\emptyset).$
- for any derived fact reachable $(c_1, c_2) \in T_P^{\omega}(\emptyset)$ , characterize the least i such that reachable $(c_1, c_2) \in T_P^i(\emptyset)$ .

# **10.2.2 Model-Theoretic Characterization: Minimal Model**

• Note: simple "Datalog" usually means "positive Datalog"

# **Definition 10.5**

For a (positive) Datalog program P, the minimal model is defined as the smallest Herbrand interpretation (wrt.  $\leq$  as in Def. 10.4) that is a model of P.

# Theorem 10.2

For a positive Datalog program P and its minimal model M, for all ground atoms  $p(c_1, \ldots, c_n)$ :

- $\mathcal{M} \models p(c_1, \ldots, c_n) \Leftrightarrow p(c_1, \ldots, c_n) \in T_P^{\omega}$ .
- $\mathcal{M} \models p(c_1, \ldots, c_n)$  if and only if for all models S of P,  $S \models p(c_1, \ldots, c_n)$ .

(recall: |= denotes the models-relation from First Order Logic)

# **Proposition 10.6**

The minimal model  $\mathcal{M}$  of a (positive) Datalog program P is the intersection of all models (i.e., models wrt. First Order Logic model theory) of P.

Proof: same as for Proposition 10.5.

578

# Non-minimal Models

Let *P* a positive Datalog program with minimal model  $\mathcal{M} = T_P^{\omega} = T_P^{\omega}(\emptyset)$ , and  $q \notin \mathcal{M}$  some ground atom.

• there exists a model  $\mathcal{M}'$  of P that makes q true.

(i.e., a positive program cannot force anything to be false; there is only "negation by default").

- Recall Slide 570: starting with q, i.e.,  $T_P^{\omega}(\{q\})$  is not appropriate (it might forget q, or even run into an alternating sequence).
- Compute  $\mathcal{M}'' = T^{\omega}_{P \cup \{q\}} = T^{\omega}_{P \cup \{q\}}(\emptyset)$  to obtain the solution, which is the minimal model of  $P \cup \{q\}$ .

• For Example 10.1, 
$$T^{\omega}_{P \cup \{q(b)\}} = \mathcal{M} \cup \{q(b), r(b)\}.$$

## Some comments on Negation

- Negative Literals:
  - The minimal model implements the *Closed-World-Assumption (CWA)*: any atom that is not contained or implied by *P* is assumed not to hold.
  - For the minimal model  $\mathcal{M}$ ,
    - if a ground atom is not in  $\mathcal{M}$ , i.e.,  $\mathcal{M} \models \neg p(a_1, \ldots, a_n)$ , classical FOL semantics (open-world) does *not* entail that  $P \models_{FOL} \neg p(a_1, \ldots, a_n)$ .

Note that  $P \models_{FOL} \neg p(a_1, \ldots, a_n)$  does not hold for any ground atom – from a positive program P no negative statements are entailed at all under FOL semantics.

- this coincides with the SQL semantics "WHERE NOT EXISTS ....".
- Negative literals in rule bodies:
  - The  $T_P$  evaluation is not applicable for rules with negation in the body.
  - Consider the previous example extended by the rule { unreachable $(X, Y) \leftarrow \text{country}(X) \land \text{country}(Y) \land \neg \text{reachable}(X, Y).$  }. How would the  $T_P$  evaluation proceed for it?
- derivation of negative facts/negative facts in rule heads: not applicable since CWA assumes all negative facts that are consistent with P ("negation by default")

580

# **10.2.3 Proof-Theoretic Approach: Resolution Calculus**

Given: a positive Datalog program PQuestion: does  $p(c_1, \ldots, c_n)$  hold?

• bottom-up computation of  $T_P$  provides a *correct* and *complete* (wrt. the minimal model) procedure for checking if some *fact* holds in the minimal model.

Every atom that is true in the minimal model has a "proof history" (tree) via the rules and facts that have been used for deriving it.

# **GENERAL RESOLUTION CALCULUS**

- an Inference System.
- a *clause* is a set of literals (semantics: disjunctive). Clause resolution takes two clauses that contain contradictory literals:

 $\ell_{1} \vee \ldots \vee \boxed{\ell_{i}} \vee \ldots \vee \ell_{k} \quad , \quad \ell_{k+1} \vee \ldots \vee \boxed{\neg \ell_{k+j}} \vee \ldots \vee \ell_{k+m} \quad , \quad \boxed{\sigma(\ell_{i}) = \sigma(\ell_{k+j})}$  $\sigma(\ell_{1} \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_{k} \vee \ell_{k+1} \vee \ldots \vee \ell_{k+j-1} \vee \ell_{k+j+1} \vee \ldots \vee \ell_{k+m})$ 

rules of the form

$$h(\bar{x}) \leftarrow b_1(\bar{x}) \wedge b_2(\bar{x}) \wedge \ldots \wedge b_n(\bar{x})$$

are equivalent to *Horn Clauses* (named after the logician Alfred Horn)

 $h(\bar{x}) \lor \neg b_1(\bar{x}) \lor \neg b_2(\bar{x}) \lor \ldots \lor \neg b_n(\bar{x})$ 

(Disjunction with only one positive literal).

582

# **ASIDE: GENERAL RESOLUTION CALCULUS: COMMENTS AND EXAMPLES**

- Tableau calculus:
  - one rule for each FOL construct ( $\land$ ,  $\lor$ ,  $\forall$ ,  $\exists$ , and the closure rule as the rule for  $\neg$ ).
  - applicable to all kinds of FOL formulas.
  - $\Rightarrow$  intuitive, very general, but a high number of possible expansions in each step.
- Resolution calculus:
  - only a single inference rule,
  - applicable to a set of (arbitrary) disjunctions.
- Any FOL formula  $\phi$  can be translated as follows:
  - Prenex Normal Form: pull quantifiers in front ("prefix"):  $\forall a, b \exists c, d \forall e \ldots : \phi'$  where  $\phi'$  is quantifier-free ("matrix"),
  - transform the matrix into conjunctive normal form (i.e., a conjunction of disjunctions of literals).
  - $\Rightarrow$  resolution calculus has the same expressiveness as tableau calculus.
  - it is intuitive, if a problem has a natural representation as a set of disjunctions.

## Disjunctive Reasoning: Sudoku

Typical Sudoku situation: "cell  $(x, y_1)$  is either 2 or 7, cell  $(x, y_2)$  is either 2 or 6, so the "2" can only be in one of them, there is 7 in  $(x, y_1)$  or 6 in  $(x, y_2)$ . As 6 is already in  $(x_2, y_2)$ , the 2 must be in  $(x, y_2)$ , and the 7 must be in  $(x, y_1)$ ."



Consider the following example (sudoku taken from (german) wikipedia):

- 3-ary predicate p ("position"), e.g. for B9: p(b, 9, 3):
- exclusion clause patterns like:  $\{\neg p(x_1,y,n), \neg p(x_2,y,n), x_1 = x_2\}$  for rows; analogously for columns and for subsquares.
- H2: must be 3 (all other numbers are already present in column H, in row 2 or in the lower right subsquare).
- A2: 2 or 3 or 7.; B2: 2 or 7 or 8.
- C2: 2 or 3 or 7.
- $\Rightarrow$  A2 and C2: 2 or 7  $\Rightarrow$  B2: 8

584

## Sudoku (cont'd)

- Query:  $answer(X, Y, N) \leftarrow p(X, Y, N)$ .
- Whenever the empty clause can be derived, an answer is given by the applied substitutions. E.g. for an already known cell,  $\{\neg p(X, Y, N)\}$  with  $\{p(b, 9, 3\}$  and  $\sigma = \{X \leftarrow b, Y \leftarrow 9, N \leftarrow 3\}$  yields the first solution.
- Simple cases like H2 (must be 3):

$$\{p(h, 2, 1), p(h, 2, 2), p(h, 2, 3), p(h, 2, 4), p(h, 2, 5), p(h, 2, 6), p(h, 2, 7), p(h, 2, 8), p(h, 2, 9)\}, \{p(e, 2, 1)\}, \{\neg p(e, 2, 1), \neg p(h, 2, 1)\}$$

$$\{p(h, 2, 2), p(h, 2, 3), p(h, 2, 4), p(h, 2, 5), p(h, 2, 6), p(h, 2, 7), p(h, 2, 8), p(h, 2, 9)\},$$

$$\vdots$$

$$\{p(h, 2, 3)\} \text{ which closes with the negated query } \{\neg p(X, Y, N)\}$$

- analogous reduction for cells A2, B2, C2:
  - $\{p(a, 2, 2), p(a, 2, 3), p(a, 2, 7)\}$ , with  $\{\neg p(a, 2, 3), \neg p(h, 2, 3)\}$  and  $\{p(h, 2, 3)\}$  to  $\{p(a, 2, 2), p(a, 2, 7)\}$ ;
  - $\{p(b,2,2), p(b,2,7), p(b,2,8)\},\$
  - $\{p(c, 2, 2), p(c, 2, 3), p(c, 2, 7)\}$  analogously to  $\{p(c, 2, 2), p(c, 2, 7)\}$

## Sudoku (cont'd)

- Situation
  - {p(a, 2, 2), p(a, 2, 7)};
  - {p(b, 2, 2), p(b, 2, 7), p(b, 2, 8)},
  - $\{p(c, 2, 2), p(c, 2, 7)\}$
- Both A2 and C2 are 2 or 7
- B2 (2, 7, or 8) must be 8
- no direct conclusion possible ...
- note: resolving to clauses with 2 literals usually yields two literals:
  - $\{a,b\}$  with  $\{\neg b,c\}$  yields  $\{a,c\}$ .

Unary clauses can be derived by matches like

 $\{a, b\}$  with  $\{a, \neg b\}$  yields  $\{a\}$ .

- $\Rightarrow$  not only clauses that are connected by a pair of contradictory literals are interesting, but also clauses that contain the same literals can be useful.
- $\Rightarrow$  a resolution reasoner maintains a connection graph for choosing its strategy.
  - human reasoners must have a plan how to proceed ...

586

# Sudoku (cont'd: Example proof for the contents of cell B2)

- write xyn for p(x, y, n):
  - this is not only a notational shortcut, but also a mapping to Boolean Logic:
  - the second line is assumed to contain all ground instances of the exclusion clause (cf. Slide 584) stating which cells must not have the same value.
     Note: the smodels tool for stable models is based on the same idea of creating all ground instances and running boolean Model Checking.



## General (FOL) Resolution Calculus

- Recall: open-world, with explicit negative literals.
- · there are always multiple possibilities to choose pairs of clauses to be resolved
- $\Rightarrow$  proof search strategy?
  - ... not the right thing for deductive databases (closed-world-assumption, equivalence to the relational algebra and SQL),
  - ... but a good basis ...
  - ... go back first to consider positive rules as a special case of disjunction *head*  $\lor \neg$  *body*.

588

# **RESOLUTION CALCULUS FOR (POSITIVE) RULES**

- a derivation rule  $head(\bar{x}) \leftarrow b_1(\bar{x}) \land b_2(\bar{x}) \land \ldots \land b_n(\bar{x})$  is equivalent to  $\neg b_1(\bar{x}) \lor \neg b_2(\bar{x}) \lor \ldots \lor \neg b_n(\bar{x}) \lor head(\bar{x})$ , or, written as a Horn clause,  $\{\neg b_1(\bar{x}), \neg b_2(\bar{x}), \ldots, \neg b_n(\bar{x}), head(\bar{x})\}$
- such a Horn clause can be seen as a *directed* disjunction with a single distinguished positive (head) literal.
- a fact  $p(\bar{c})$  corresponds to a unary clause consisting of a single positive literal  $\{p(\bar{c})\}$ .

# Bottom-Up: Resolution as Forward Reasoning

Example:

```
Consider the rule subordinate(x, y) \leftarrow \text{works-for}(x, d) \land \text{manages}(y, d)
(forget about x \neq y for now)
```

The corresponding clause is

```
{subordinate(X, Y), ¬works-for(X, D), ¬manages(Y, D)}.
```

Consider the (unary) fact clauses {works-for(mary,sales)} and {manages(alice,sales)}.



... derives subordinate(mary,alice).

• obviously, for every ground atom  $p(a_1, \ldots, a_n)$ ,  $P \vdash_{\mathsf{Res}} p(a_1, \ldots, a_n)$  if and only if  $p(a_1, \ldots, a_n) \in T_P^{\omega}$ .

590

# **TOP-DOWN: RESOLUTION CALCULUS AS BACKWARD REASONING**

- used in PROLOG systems:
   SLD Resolution (Selection-Rule-Driven Linear Resolution for Definite Clauses)
- given: a "program" *P* of rules and facts, and a claimed fact  $answer(\bar{c})$ . Show:  $P \models answer(\bar{c})$ ?
- Resolution as a refutation strategy: prove that  $\neg answer(\bar{c})$  is inconsistent with *P*.
- a negated atom can be refuted if it matches the head of a rule and all of the body atoms of the rule can be proven. Apply recursively:
  - get a new "goal clause" (i.e., a clause containing only negative literals)
     [⇒] linear proof;
  - note that multiple rule heads can match (SLD: first rule first);
  - note that multiple literals can match: resolve literals from left to right (i.e., depth-first).
- try to derive the empty (goal) clause: then it is shown that P ∪ {¬answer(c̄)} is unsatisfiable, i.e., P ⊨ answer(c̄).



592

# **SLD RESOLUTION FOR ANSWERS**

- the initial goal (=query) contains free variables,
- · collect the union/concatenation of all substitutions applied
- if the empty clause is derived, the restriction of the resulting substitution to the variables in the query is the *answer substitution*.
- do backtracking (alternative closing substitutions with other facts, alternative rules with the same head),
- compute further answers.

# SLD RESOLUTION WITH ANSWERS: EXAMPLE

"All organizations that have their headquarters in the capital of a European member country with more than 10000000 inhabitants"



Resolve  $C_1$  with  $C_4$  (the only rule that matches) by  $\sigma_1 : \{O \to X\}$ :

 $C_5: \{\neg \mathsf{eBC}(C), \neg \mathsf{isM}(C, X, \_), \neg \mathsf{hC}(X, C)\}.$ 

594

Resolve  $C_5$  with  $C_2$  (first literal):

 $C_6: \ \{\neg \mathsf{enc}(C, ``\mathsf{Europe"}, \_), \neg \mathsf{c}(\_, C, \_, \_, P), \neg P > 10000000, \neg \mathsf{isM}(C, X, \_), \neg \mathsf{hC}(X, C)\} \ .$ 

Resolve  $C_6$  with fact enc("B", "Europe", 100) (one out of many candidates) by  $\sigma_2 : \{C \to "B"\}$ :

 $C_7: \{\neg c(\_, "B", \_, \_, P), \neg P > 1000000, \neg \mathsf{isM}("B", X, \_), \neg \mathsf{hC}(X, "B")\}.$ 

Resolve with fact c("Belgium", "B", "Brussels", "Brabant", \_, 10170241) by  $\sigma_6 : \{P \rightarrow 10170241\}$  and remove the (false) instantiated literal  $\neg 10170241 > 10000000$ :

 $C_8: \{ \neg \mathsf{isM}(\mathsf{"B"}, X, \_), \neg \mathsf{hC}(X, \mathsf{"B"}) \} .$ 

Resolve with fact isM("B","EU","member") by  $\sigma_4 : \{X \rightarrow "EU"\}$ :

 $C_9: \{\neg hC("EU", "B")\}.$ 

Resolve with  $C_3$  by  $\sigma_5 : \{ O \rightarrow \text{``EU"}, C \rightarrow \text{``B"} \}$ :

 $C_{10}: \ \{\neg \mathsf{c}(\_, ``\mathsf{B"}, Cap, CapP, \_, \_), \neg \mathsf{org}(``\mathsf{EU"}, \_, Cap, ``\mathsf{B"}, CapP, \_)\} \ .$ 

Resolve with fact c("Belgium", "B", "Brussels", "Brabant", \_, \_):

 $C_{11}: \{\neg org("EU", "Brussels", "B", "Brabant", _)\}$ .

Resolve with fact org("EU", \_, "Brussels", "B", "Brabant", \_) and obtain the empty clause.

This generates the first answer X/ "EU".

Backtracking ... resolve  $C_8$  with fact isM("B", "UN") to obtain

 $C_{11}: \{\neg hC("UN", "B")\}$ .

Resolve again with  $C_3$  by  $\{O \rightarrow \text{``UN''}, C \rightarrow \text{``B''}\}$  and continue as above. The empty clause cannot be derived (the headquarters of the UN are in New York). Backtrack again, resolve  $C_8$  with NATO, return X/``NATO'', analogously check all organizations where Belgium is a member, and return all organizations located in Brussels.

Backtracking then to  $C_5$ , try the next european country etc.



596

# 10.3 Aside: Full Prolog

- allows function symbols
- not just matching, but *unification* of terms (that contain variables someewhere): p(f(X, g(Y))) unifies with  $\neg p(f(h(Z), Z))$  via  $\sigma = \{Z \rightarrow g(Y), X \rightarrow h(g(Y))\}.$
- derives the empty clause *and* an answer substitution e.g. when asking ?-subordinate(*X*,alice).

X/mary X/bob

- uses backtracking:
  - if search for an answer is not successful, try another way,
  - if an answer is found, report it and try another way (next substitution, next rule),
  - generates a proof search tree.
- Prolog *Programming* goes even further: "cut" and "fail" to control the exploration of the search space.

Then, the order of rules and literals becomes extremely important.

Aside: Prolog Programming: Cut

The "cut" predicate (written as "!") fixes the bindings up to that literal and does not search for other proofs (e.g., for alternative bindings for existential variables):

- $F(C) \equiv \exists CN, Cap, CapP, A, Pop : \mathsf{country}(CN, C, Cap, CapP, A, Pop) \land \\ \exists Org, Abbr, Est, T : \mathsf{organization}(Org, Abbr, Cap, C, CapP, Est), \mathsf{isMember}(C, Org, T) \end{cases}$ 
  - ?-res(C) returns every result country several times for each organization that has its city in the capital.
  - ?-res2(C) returns every result country only once, since there is no backtracking in rule hqInCap2 that would cause to search other proofs for e.g. hqInCap2('B')

```
:- include(mondial).
```

```
res(C) :- country(_,C,Cap,CapP,_,_), hqInCap(C,Cap,CapP).
```

```
hqInCap(C,Cap,CapP) :- organization(Org,_,Cap,C,CapP,_), isMember(C,Org,_).
```

```
res2(C) :- country(_,C,Cap,CapP,_,_), hqInCap2(C,Cap,CapP).
```

hqInCap2(C,Cap,CapP) :- organization(Org,\_,Cap,C,CapP,\_), isMember(C,Org,\_),!.

[Filename: Datalog/prologcut.P]

- cut san serve –declaratively– as a SQL DISTINCT
- in combination with EXISTS (each existing thing would otherwise be checked),
- and for optimization of traversing proof trees.

598

Aside: Prolog Programming: Output

- the "cut" predicate fixes the bindings up to that literal and does not search for other proofs:
- write any term to stdout with write(term),
- the nl predicate outputs a newline to stdout.
- tell me, when Paris is investigated ...

```
:- include(mondial).
```

[Filename: Datalog/prologparis.P]

- in check(X,Y), Y must be bound upon calling it (XSB warns) the rules are not safe,
- but "safe" is a bottom-up Datalog issue, in Prolog Programming, such unsafe procedural rules are common. (when called, the variables are already bound from atoms evaluated before)

# Aside: Prolog Programming: Input and fail

- read(X) reads a term. The input must be finished by a ".".
- predicate fail is used when Prolog should "execute" the rule as "proof search" to do something, and then ... fail:
- below, shouldI fails if "N" is input. Then, also shouldICheck(X) fails, and the body for res(C) is not satisfied for this C. Try next C.

```
600
```

## Aside: Prolog Exercise

Consider again the program prologask.P from the previous slide.

When running it, the output "here ... country is still ..." when it is actually already finished with the respective country, demonstrates that useless work is done.

Where to place a cut to avoid this?

## Aside: Prolog Documentation

• see XSB Manual Part I, Section 6 "Standard Predicates and Predicates of General Use".

# **10.4 Positive Recursive Datalog**

• a Datalog Program is called *recursive* if ...

**Dependency Graph** 

## **Definition 10.6**

For a positive Datalog program *P* over a (relational) signature  $\mathbf{R} = \{R_1, \dots, R_n\}$ , its Dependency Graph G = (V, E) is defined as follows:

- $V = \{R_1, \ldots, R_n\}$  is the set of vertexes,
- *R<sub>i</sub>* → *R<sub>j</sub>* ∈ *E* if *P* contains a rule with head predicate *R<sub>j</sub>* and *R<sub>i</sub>* occurs in its body ("*R<sub>j</sub>* depends on *R<sub>i</sub>*").

# **Definition 10.7**

A Datalog program is called recursive if its dependency graph contains a cycle.

602

#### Consequences

... the definitions up to now hold for nonrecursive programs and for recursive ones:

- the minimal model is defined as usual,
- $T_P$  and  $T_P^{\omega}$  are defined as usual,
- the resolution proofs exist.
  - Systems based on PROLOG's SLD resolution potentially run into infinite proof search trees
    - (can be blocked by (expensive) bookkeeping)
  - XSB supports "tabling" which makes it more efficient and prevents it from infinite loops (tabling stores derived facts for reuse),
  - must be activated (see below).

## Example: Transitive Closure

- $tc(x,y) \leftarrow p(x,y)$ .  $tc(x,y) \leftarrow \exists z: tc(x,z) \land tc(z,y)$ .
- XSB: % as comment sign,
- :- auto\_table. for activating automatic tabling,
- manual tabling can be switched on with
  - :- table  $R_1/k_1$ , ...,  $R_n/k_n$ . for  $k_i$ -ary table  $R_i$ .

% :- auto\_table.

```
:- table borders/3, reachable/2.
:- include(mondial).
borders(Y,X,Z) :- borders(X,Y,Z). % make it symmetric.
reachable(X,Y) :- borders(X,Y,_).
reachable(X,Y) :- reachable(X,Z), borders(Z,Y,_).
```

[Filename: Datalog/transitiveclosure.P]

# Exercise

Complete the program from Slide 407 such that it also includes rivers flowing through lakes into others.

```
604
```

Additional Syntax, Built-Ins

- arithmetic operations: + \* /
- assignment by var is term in the body
- comparisons: as usual,  $\setminus$ = for  $\neq$ , =< and >= for  $\leq$  and  $\geq$ .
- see also XSB Manual Part I Sections 3.10.5 (Inline Predicates) and 4.3 (Operators).

```
:- include(mondial).
cview(N,C,Pop,A,Density) :- country(N,C,_,_,A,Pop), Density is Pop/A.
[Filename: Datalog/arithmetics.P]
```

# 10.5 Datalog with Negation

Consider conjunctive queries that include *negative* Literals.

• e.g.  $F(C) = \exists CN, Cap, CapP, A, Pop$ :

 $(\mathsf{country}(CN, C, Cap, CapP, A, Pop) \land \neg\mathsf{ismember}(C, \mathsf{``EU"}, \mathsf{``member"}))$ 

- the database contains only positive facts, so no negative information can be logically implied!
- SQL:

```
SELECT code FROM country
WHERE NOT (code,'EU','member') IN (SELECT * FROM ismember);
yields 214 results.
```

- Databases: "Closed World Semantics" tuples that are not stored are assumed not to hold.
- $\Rightarrow$  database query semantics deviates from standard FOL model theory.
- $\Rightarrow$  a different model theory applies!

606

# Closed World/Default Negation

- actually known + used in SQL without problems,
- the idea of the Minimal Model is analogous: everything that cannot be proven is false in the Minimal Model.
  - But the *Minimal Model is not well-defined in presence of negation:* Consider P = {p ← ¬q}: Both M<sub>1</sub> = {p} and M<sub>2</sub> = {q} are minimal models of P.
- Prolog: SLD-resolution extended to SLD-NF-resolution:
  - NF: Negation (of  $p(c_1, ..., c_n)$ ) as "(finite) failure" to prove  $p(c_1, ..., c_n)$  (only for ground atoms; cf. safety):
  - Open a resolution proof for  $\neg p(c_1, \ldots, c_n)$  as usual and show that after finitely many steps there is no more progress towards the empty clause.
  - Example: For P = {p ← ¬q}, SLD-NF for ?- p starts a proof for the body, i.e., for ¬q which fails (the rule is equivalent to the clause {p,q}) immediately.
     Thus ¬q is "proven" and p is confirmed the answer to ?- p is "yes".
- Preview: both {p ← ¬q} and {q ← ¬p} are logically equivalent to p ∨ q, but, as programs, have different semantics!

# **NEGATION IN THE BODY: DATALOG**<sup>¬</sup>

The language Datalog<sup>¬</sup> extends positive Datalog as follows:

 the rule body is allowed to contain also negative literals: Rules are now of the form

$$H \leftarrow L_1 \land \ldots \land L_k$$

where each  $L_i$  is a positive  $(p(a_1, \ldots, a_n))$  or negative  $(\neg p(a_1, \ldots, a_m))$  literal.

• Safety requirement: every variable that occurs in a negative literal must also occur in a positive one before, e.g.

unreachable(X, Y)  $\leftarrow$  country(X)  $\land$  country(Y)  $\land \neg$ reachable(X, Y).

608

#### **Formal Semantics**

The  $T_P$  operator (cf. Slide 568) is extended as follows:

For a set I of ground atoms,

$$T_P(I) := \{ \sigma(H): \sigma \text{ is a ground substitution and there is a rule} \\ H \leftarrow L_1 \land \ldots \land L_k \text{ in } P \text{ such that for each } i = 1..k \\ \sigma(p_i(\bar{a})) \in I \text{ if } L_i = p_i(\bar{a}) \text{ is positive,} \\ \sigma(p_i(\bar{a})) \notin I \text{ if } L_i = \neg p_i(\bar{a}) \text{ is negative } \}$$

- The plain  $T_P^{\omega}$  computation is not suitable: In the first "round" things are false that will become true later
- $\Rightarrow$  "wait" before evaluating a negative literal  $\neg p(c_1, \ldots, c_n)$  until the predicate p is completely computed.

(note: SLD resolution does automatically "stratify" when it opens the subproof for  $\neg p(c_1, \ldots, c_n)$  and tries to complete it with the rules for *p*.)

# STRATIFICATION

## Dependency Graph with Negation

Extend Definition 10.6:

## **Definition 10.8**

For a Datalog<sup>¬</sup> program *P* over a (relational) signature  $\mathbf{R} = \{R_1, \dots, R_n\}$ , its Dependency Graph G = (V, E) is defined as follows:

- $V = \{R_1, \ldots, R_n\}$  is the set of vertexes,
- *R<sub>i</sub>* → *R<sub>j</sub>* ∈ *E* if *P* contains a rule with head predicate *R<sub>j</sub>* and *R<sub>i</sub>* occurs positively in its body ("*R<sub>j</sub>* depends positively on *R<sub>i</sub>*").
- *R<sub>i</sub>* → *R<sub>j</sub>* ∈ *E* if *P* contains a rule with head predicate *R<sub>j</sub>* and *R<sub>i</sub>* occurs negatively in its body ("*R<sub>j</sub>* depends negatively on *R<sub>i</sub>*").

If the dependency graph does not contain a negative cycle (i.e., a cycle where *at least* one edge is negative) then there exists a simple, intuitive semantics (note that positive cycles are allowed).

610

#### Stratification

#### **Definition 10.9**

Given a Datalog<sup>¬</sup> program *P* without negative cycles over a signature  $\Sigma$ , a stratification is a partitioning of  $\Sigma$  into strata  $S_1, \ldots, S_n$  by a stratification mapping  $\sigma : \Sigma \to \{1, \ldots, n\}$  such that

- if p depends positively on q, then  $\sigma(p) \ge \sigma(q)$ ,
- if p depends negatively on q, then  $\sigma(p) > \sigma(q)$ ,
- *if such a stratification is possible, P is called* stratifiable.

Define  $P_i$  to be the set of rules in P whose head predicate is in  $S_i$ .

#### **Properties**

- $S_1$ : predicate symbols (incl. facts) that do not depend negatively on any other predicate,
- $S_i$ : predicate symbols that depend positively only on predicate symbols in  $S_0, \ldots, S_i$ ,
- $S_i$ : predicate symbols that depend negatively only on predicate symbols in  $S_0, \ldots, S_{i-1}$ .
- predicates that are positively cyclic dependent on each other belong to the same stratum.
- $\{P_1, \ldots, P_n\}$  is a partitioning of *P*.

Note: there may be several stratifications of a program (any partitioning that is compatible with the priority order given by the negative dependencies).

#### Stratification

#### **Proposition 10.7**

- every nonrecursive Datalog<sup>¬</sup> program is stratifiable,
- many recursive Datalog<sup>¬</sup> programs are also stratifiable. (cf. reachable, non-reachable)

612

П

# STRATIFIED MODEL

Stratification allows to compute a model incrementally (bottom-up): Compute each stratum by "freezing" the IDB predicates defined in the previous stratum like EDB relations/facts:

#### **Definition 10.10**

Let  $P = \{P_1, P_2, \dots, P_n\}$  be a stratified program. Then, S(P) defined as follows is the stratified model of P:

$$\begin{aligned} \mathcal{I}_0 &= \emptyset \\ \mathcal{I}_k &= T^{\omega}_{P_k \cup \mathcal{I}_{k-1}}(\emptyset) \quad \textit{for } 1 \leq k \leq n \\ \mathcal{S}(P) &= \mathcal{I}_n \end{aligned}$$

(with every  $P_i$  a set of rules and every  $\mathcal{I}_i$  a set of ground atoms,  $P_k \cup \mathcal{I}_{k-1}$  is a Datalog program that fits into stratum  $\mathcal{S}_k$ ).

#### **Proposition 10.8**

- $\mathcal{S}(P)$  does not depend on the chosen stratification,
- $\mathcal{S}(P)$  is a model of P,
- $\mathcal{S}(P)$  is minimal (i.e., no  $\mathcal{M}' \subsetneq \mathcal{S}(P)$  is a model of P),
- for programs containing negation, there are in general several models that are minimal.  $_{\Box}$

#### Comments

- bottom-up stratified evaluation is the counterpart to top-down SLD-NF evaluation,
- · tabling fits well with stratification,
- XSB does stratification automatically if a program contains negation.

#### Exercise

Prove that Definition 10.10 is equivalent to the following characterization:

$$\mathcal{J}_0 = \emptyset$$
  

$$\mathcal{J}_k = T^{\omega}_{P_1 \cup \dots \cup P_k}(\mathcal{J}_{k-1}) \text{ for } 1 \le k \le n$$
  

$$\mathcal{S}'(P) = \mathcal{J}_n$$

614

#### Monotonic vs. Nonmonotonic Reasoning

#### Definition 10.11

For a given set of input formulas  $\phi$ , and a reasoning mechanism M, let  $Th_M(\phi)$  denote the "theory of  $\phi$  wrt. M", i.e., the set of conclusions  $\psi$  such that  $\phi \models_M \psi$ .

A reasoning mechanism M is monotonic if

$$\phi_1 \subseteq \phi_2 \Rightarrow Th_M(\phi_1) \subseteq Th_M(\phi_2)$$

- · FOL is monotonic,
- The Minimal Model semantics is monotonic,
- Default Logic and Human Reasoning is nonmonotonic (allowing conclusions in presence of incomplete knowledge that can be revised upon additional information),
- Stratified semantics is nonmonotonic.

#### Exercise

- Give an example for the nonmonotonicity of the stratified semantics,
- show that for a stratifiable program P there can be multiple minimal models.

# 10.5.1 (Stratified) Nonrecursive Datalog with Negation vs. Relational Algebra and SQL

## Theorem 10.3

Nonrecursive Datalog with (stratified) negation with a single-predicate result is equivalent to the relational algebra.  $\Box$ 

- Means: every nonrecursive Datalog<sup>¬</sup> program that defines a single *n*-ary result predicate res/*n* can be expressed by a calculus query with *n* free variables, and equivalently by a relational algebra expression with an *N*-ary result relation, and
- every *n*-ary relational algebra expression can be expressed by a nonrecursive Datalog<sup>¬</sup> program that defines a single *n*-ary result predicate res/*n*.

#### Exercise:

- prove the "Algebra  $\rightarrow$  Datalog" direction (by structural induction).
- Given a (safe) rule H ← C<sub>1</sub> ∧ ... ∧ C<sub>n</sub> ∧ D<sub>n+1</sub> ∧ ... ∧ D<sub>n+k</sub> where the C<sub>i</sub> are positive literals and the D<sub>i</sub> are negative literals, give a relational algebra expression that returns the relation defined by it.

```
616
```

## Example: Relational Division

• recall: the relational division is defined in the relational algebra by two negations

Organizations that have at least one member on each continent:

[Filename: Datalog/orgOnContsDiv.P]

• note: call of the PROLOG standard predicate

```
?- findall(_0, result(_0), L).
```

returns all answers as a PROLOG list.

• compare with expressing this query in SQL.

# PROLOG FINDALL

Syntax:

findall(variable, predicate(many variables), listvariable)

- the *variable* must be bound in the predicate query; all other variables in the predicate query are local to it,
- · listvariable does not occur in the predicate query.

```
?- findall(A,continent(N,A),L).
```

```
A = _h44
N = _h66
L = [9562488,45095292,8503474,30254708,39872000]
?- findall(N,city(N,'D',P,Pop,La,Lo,El),L).
```

% all names of german cities.

Large lists sometimes lead to a crash:

```
?- findall(Pop,city(N,C,P,Pop,La,Lo,El),L).
```

618

# AGGREGATION

- example see next slide.
- PROLOG dialects supports aggregation
- XSB: via PROLOG collections:
  - collect values in a bag:

```
\mathsf{bagof}(var_1, var_2 \ \ \cdots \ \ var_n \ \ pred(var_1, \ldots, var_n), collvar)
```

```
for collvar := bagof\{var_1 \mid \exists var_2, \dots, var_n : pred(var_1, \dots, var_n)\}
```

- explicitly program the aggregation operator recursively over the collection.
- collection is a PROLOG list organized as head, tail: Syntax: [H|T] or . (H,T), empty list is [].
- Note: aggregation operations also require stratification the predicates used in the subquery must be computed before.

```
:- include(mondial).
citypops(C,B) :- bagof(Pop,N^P^Lo^La^El^city(N,C,P,Pop,La,Lo,El),B).
% citypops('A',L).
% L = [1583000,10102,87321,null,144000,203000,118000,238000,51102]
% citypops('A',.(H,T)).
% H = 1583000
% T = [10102,87321,null,144000,203000,118000,238000,51102]
sum(X,[H|T]) := sum(Y,T), H = null, Y = null, X is H + Y.
sum(H,[H|T]) := sum(null,T), H = null.
sum(X, [null|T]) :- sum(X,T).
sum(null,[]).
% Test: ?- sum(N,[1,2,3,4,5]). yields 15
citypopsum(C,X) :- citypops(C,B), sum(X,B).
% citypopsum('A',X).
% X = 2434525
                                     [Filename: Datalog/aggregation.P]
```

```
620
```

Demonstrate both collection syntaxes:

```
:- include(mondial).
citypops(C,B) :- bagof(Pop,N^P^Lo^La^El^city(N,C,P,Pop,La,Lo,El),B).
sum1(X,[H|T]) :- sum1(Y,T), H \= null, Y \= null, X is H + Y.
sum1(H,[H|T]) :- sum1(null,T), H \= null.
sum1(X,[null|T]) :- sum1(X,T).
sum1(null,[]).
sum2(X,.(H,T)) :- sum2(Y,T), H \= null, Y \= null, X is H + Y.
sum2(H,.(H,T)) :- sum2(null,T), H \= null.
sum2(X,.(null,T)) :- sum2(X,T).
sum2(null,[]).
citypopsum(C,X,Y) :- citypops(C,B), sum1(X,B), sum2(Y,B).
[Filename: Datalog/aggregation2.P]
```

Aside: Tabling with Answer Subsumption

- XSB Documentation, Section 5.4
- tabling with subsumption: "subsumed" (wrt. some ordering) answers are not stored
- $\Rightarrow$  only "maximal" ones remain.

```
:- include(mondial).
:- table citypopmax(_,po(> /2)). %% blank before "/" is important!
citypopmax(C,N) :- city(_,C,_,N,_,_,), N \= null.
?- citypopmax('D',P).
```

[Filename: Datalog/aggrsubsumpt.P]

- works only for min/max, not count/sum (these are not idempotent)
- · see documentation: shortest paths

622

# 10.5.2 (Stratified) Recursive Datalog with Negation

- The stratified semantics seamlessly covers stratifiable *recursive* Datalog<sup>¬</sup> programs.
- expressiveness covers Algebra/Calculus + Recursion.

```
% :- auto_table.
:- table borders/3, reachable/2.
:- include(mondial).
borders(Y,X,Z) :- borders(X,Y,Z). % make it symmetric.
reachable(X,Y) :- borders(X,Y,_).
reachable(X,Y) :- reachable(X,Z), borders(Z,Y,_).
notReachable(X,Y) :- country(_,X,_,_,_), country(_,Y,_,_,_),
not reachable(X,Y).
```

[Filename: Datalog/transitiveclosure2.P]

#### Exercise

• Give the intermediate steps of the  $T_P^{\omega}$ -based stratified evaluation for the above program.

## Summary

- bottom-up inefficient when regarding a single query.
- IDB predicates can be seen as views:
  - materialization of views not unusual in DB (when frequently used, seldomly changing)
  - view maintenance strategies (upon updates of underlying tables) in LP exist:
  - seminaive evaluation of  $T_P^n$ : consider only rule instantiations where at least one atom has been derived in the previous round for computing the next one.
- tabling is already a mixture between bottom-up and top-down.
- data transformation/integration applications:
  - transform the whole input database(s),
  - export certain IDB relations as "resulting database",
  - (e.g. generation of the MONDIAL database from Web sources with F-Logic in 1998).

624

#### Summary: Expressiveness of Datalog

- · negation in the body restricted to stratifiable knowledge bases
- no existentials

note: it is e.g. not possible to express that every country has a capital if not all of them are explicitly known.

Datalog is a database language, not an ontology language.

 $\Rightarrow$  Semantic Web uses different languages.

- no disjunction in the head  $P \rightarrow (Q \lor R)$
- unique name assumption, no equality

# Chapter 11 Datalog Knowledge Bases II

# **NEGATION IN THE BODY: CYCLIC NEGATIVE DEPENDENCIES**

A program whose dependency graph contains a *negative cycle* cannot be stratified.

• Consider the program  $P = \{p(b) \leftarrow \neg p(a)\}$  (without any assured facts). It has three models,  $\mathcal{M}_1 = \{p(b)\}, \ \mathcal{M}_2 = \{p(a)\}$ , and  $\mathcal{M}_3 = \{p(a), p(b)\}$ . Both  $\mathcal{M}_1$  and  $\mathcal{M}_2$  are minimal.

Which of the models is "preferable", given P as a knowledge base?

- well-founded semantics (still polynomial)
- stable semantics (answer set programming) (exponential)
- the rule is logically equivalent to  $p(a) \vee p(b)$  but as a rule, it can be read to have a more "directed" meaning:
  - "if p(a) cannot be shown, then assume p(b)".

626

## Example: Win-Move-Game

- 2 players,
- positions on a board that are connected by (directed) moves (relation "move(x,y)"),
- first player puts a pebble on a position,
- players alternately move the pebble from x to a connected y,
- if a player cannot move, he loses.
- Question: which positions are "winning" positions, "losing" position, or "drawn" positions?

The following program "describes" the game:

win(X) :- move(X,Y), not win(Y).

• the dependency graph contains a negative cycle:





# Well-Founded Semantics: Motivation

- ... switch from "stupid" bottom-up to well-founded argumentation "why or why not".
  - every fact has an individual finite proof (positive/existential part: linear; not-exists/forall part: multiple ((finitely) failed) subproofs)
  - but not stratified (but "dynamically stratified"/"locally stratified"/"ground-stratified")
  - 1. basic facts,
  - 2. apply rules based on existing knowledge
  - 3. additional facts,
  - 4. continue with (2);
  - 5. including "negative facts" under closed-world assumption (CWA).
  - Does this need full reasoning? (tableau proofs obviously cover it)
  - is resolution sufficient? (yes, it's only rule applications)
  - theory: how to characterize the model?
  - three-valued logic: yes-no-undefined (win-move: lost/won/drawn)
  - how to compute the model efficiently?

628

# ANALYSIS

- which atoms are definitely true?
  - the facts
  - instantiations  $\sigma(H)$  of rule heads of rules  $H \leftarrow C_1 \land \ldots \land C_n \land \neg D_1 \land \ldots \land \neg D_k$ 
    - \* where all  $\sigma(C_i)$  are definitely true, and
    - $\star\,$  where all  $\sigma(D_i)$  are definitely false.
- which atoms are definitely false (under CWA)?
  - instances of EDB predicates that are not amongst the given facts,
  - ground instances p(...) of IDB predicates such that for all rules whose rule head H unifies with p(...) as  $\sigma(H)$  (there might be several such rules with p(...) in their head):  $H \leftarrow C_1 \land \ldots \land C_n \land \neg D_1 \land \ldots \land \neg D_k$ 
    - $\star$  some  $\sigma(C_i)$  is definitely false, or
    - \* some  $\sigma(D_i)$  is definitely true.
- idea: start with nothing. Derive some definitively true things and some definitively false ones.
- based on the obtained knowledge, do "next round",
- care for "still unknown" things.

```
Well-Founded Semantics: For What
Many real problems are stratified.
Most (relational/SQL) queries are stratified.
WFS goes beyond classical queries:
many problems can be encoded in Datalog wrt. well-founded semantics
Non-Stratified examples:

logical puzzles ;)
planning problems
can_start(Y) ← completed(X), additional conditions.
argumentation contexts
holds(...) ← holds(...), ¬ holds(...), additional conditions.

Let's have a look at the theory ...
```

630

# **REDUCT OF A PROGRAM**

Consider a Herbrand interpretation (i.e., a set of ground facts)  $\mathcal{H}$ .

# Definition 11.1 (Reduct of a Program)

The reduct  $P^{\mathcal{H}}$  of a program P wrt. a Herbrand interpretation  $\mathcal{H}$  is obtained as follows:

- let  $P_g$  denote the grounding of P, i.e. the set of all ground instances of rules in P over elements of the Herbrand universe of  $\mathcal{H} \cup P$ .
- delete from  $P_g$  all rules that contain a negative literal  $\neg a$  in the body such that  $a \in \mathcal{H}$ , (these rule bodies cannot be satisfied in  $\mathcal{H}$ )

delete all remaining negative literals in the bodies of the remaining rules.
 (for those ¬a, a ∉ H, i.e., these literals are satisfied in H)

## Properties of $P^{\mathcal{H}}$

- $P^{\mathcal{H}}$  is a (ground) positive program.
- If *H* is a model of *P*, then *T*<sub>*P*<sup>*H*</sup>(*H*) ⊆ *H*.
  (note: use *T*<sub>*P*<sup>*H*</sup>(*H* here, not *T*<sup>ω</sup>, but run it on *H*)
  </sub></sub>

# 11.1 Stable Models I

# Definition 11.2 (M. Gelfond, V. Lifschitz, ICLP 1988)

A Herbrand interpretation  $\mathcal{H}$  is a stable model of a Datalog<sup>¬</sup> program P, if

$$T^{\omega}_{P\mathcal{H}}(\emptyset) = \mathcal{H}.$$

• note that a program P can have several stable models.

**Remark and Exercise** 

Note that the definition of stable models is based on  $T^{\omega}_{P\mathcal{H}}(\emptyset)$ .

Consider  $P = \{p(a) := p(a)\}$  and  $\mathcal{H} = \{p(a)\}; P^{\mathcal{H}} = P$ .  $\mathcal{H}$  is a model of P, and  $T^{\omega}_{P^{\mathcal{H}}}(\mathcal{H}) = \mathcal{H}$ .

But,  $T^{\omega}_{P^{\mathcal{H}}}(\emptyset) = \emptyset$ , i.e.,  $\mathcal{H}$  is not a stable model (p(a) is not "supported").

 $\mathcal{H}' = \{p(a), p(b), q(b)\}$  is also a model of *P*, which is also (obviously) not stable.

Obviously,  $\emptyset$  is a stable model of P – and thus, is the only one.

Note that the above example is a positive Datalog program. For positive Datalog programs P, and any  $\mathcal{H}$ ,  $P^{\mathcal{H}} = \text{ground}(P)$  (i.e., all ground instances of rules of P) and  $T^{\omega}_{\text{ground}(P)}(\emptyset) = T^{\omega}_{P}(\emptyset)$  is the only stable model.

#### Stable Models - Example

Consider the following program P:

```
q(a) :- not p(a).
```

[Filename: Datalog/qnotp.s]

Logically, the rule is equivalent to  $p(a) \lor q(a)$ .

• The program has one stable model:

```
> lparse -n 0 qnotp.s|smodels
Answer: 1
Stable Model: q(a)
True
```

For  $\mathcal{H} = \{q(a)\}, P^{\mathcal{H}} = \{q(a) := true\}$  and  $T^{\omega}_{P^{\mathcal{H}}}(\emptyset) = \{q(a)\}$ , thus  $\mathcal{H}$  is stable.

- Consider  $\mathcal{H}' = \{p(a)\}$ . It is a model of *P*.  $P^{\mathcal{H}'} = \emptyset$  and  $T^{\omega}_{P^{\mathcal{H}'}}(\emptyset) = \emptyset$ . The derivation of p(a) is "not supported" by *P*;  $\mathcal{H}'$  is not stable.
- so, in Stable Models Semantics, the rule does not mean disjunction, but is directed.

Stable Models – Example

Consider the following program:

```
q(a) :- not p(a).
p(a) :- not q(a).
[Filename: Datalog/porq.s]
```

Logically, each of the rules is equivalent to  $p(a) \lor q(a)$ .

• The program has two total stable models, and one partial (which is the well-founded model):

```
> lparse -n 0 --partial porq.s|smodels
Answer: 1
Stable Model: q(a)
Answer: 2
Stable Model: p(a)
Answer: 3
Stable Model: q'(a) p'(a)
```

- thus, both rules together represent disjunction.
- Note that  $\{p(a), q(a)\}$  is a model, but not a stable model.
- There is no possibility in Datalog<sup>¬</sup> to assert ¬q(a) to forbid one of the models. (in smodels, this will be allowed)

634

Stable Models – Example

Consider the following program:

p(a). q(a) :- not p(a). p(a) :- not q(a).

[Filename: Datalog/pporq.s]

- The program has only one stable model:  $\{p(a)\}$ .
- This model is also the well-founded model.

#### WinMove with Stable Models

· lparse does not accept don't-care-variables.



[Filename: Datalog/winmove.s]

- lparse -n 0 -d none winmove.s | smodels yields two total two-valued stable models.
- drawn cycle between h and m: once w/l, other l/w
- wfm = intersection of stable models, minimal 3-valued model.

636

Stable Models – First Summary

- A Datalog<sup>¬</sup> program may have several stable models.
- Finding the stable models of a program is exponential (optimization strategies exist)
- · come back to the well-founded semantics
  - cheaper (polynomial),
  - returns a *unique* reasonable result in cases where disjunction is not needed or not intended,
  - cf. win-move game: drawn positions are neither lost nor won.
- ... a closer investigation of stable models semantics will be given on Slides 671 ff.

# **11.2 Well-Founded Semantics**

 recall the considerations from Slides 628 ff.: well-founded non-stratified "argumentation" which facts can be derived to be true or false

#### Main Problem:

How to deal with true-unknown-false:

- model-theoretic: three-valued logic
- practically: apply a trick to be able to use the existing 2-valued  $T_P$  operator for *positive* Datalog.

#### Definition

## Definition 11.3 (A. Van Gelder, K.A. Ross, J.S. Schlipf, PODS 1988)

Given a Datalog<sup>¬</sup> program P, the well-founded model of P is the minimal 3-valued stable model of P.

- from the practical view not very promising ... not only to guess stable models, yet even 3-valued.
- have a look at this definition later.

638

# **ALTERNATING FIXPOINT COMPUTATION FOR WFS**

The Alternating Fixpoint Computation [A. Van Gelder, PODS 1989] mirrors the well-foundedness of the derivation:

## **Definition 11.4**

Given a Datalog<sup>¬</sup> program *P* over a signature  $\Sigma$ , define the sequence  $I_0, I_1, \ldots$  of Herbrand interpretations over  $\Sigma$  als follows:

$$I_0 := \emptyset$$
  
$$I_{i+1} := T_{P^{I_i}}^{\omega}(\emptyset)$$

• Does ((*I<sub>k</sub>*)) converge? No. And Yes.

 Is there a fixpoint? Yes. There are two fixpoints!

... let's have a look ...

#### Exercise

Evaluate  $((I_k))$  for the win-move example.

#### Alternating Fixpoint: Analysis

• Consider first the program  $P^{\text{facts}}$  which consists only of the facts (= fact rules) in P:

-  $T^{\omega}_{P_{\text{facts}}}(\emptyset) = T^{1}_{P_{\text{facts}}}(\emptyset)$  makes all facts true that are contained in the program.

- Consider next the program *P*<sup>+</sup> which is obtained from ground(*P*) by deleting all rules that contain any negative literal:
  - $P^+$ : corresponds to "all negative literals are false". Recall that  $\mathcal{HB}_P$  denotes the interpretation that makes all possible atoms over the Herbrand Universe of P true. With this,  $P^+ = P^{\mathcal{HB}_P}$ .
  - (P<sup>+</sup> can be equivalently obtained by first deleting all rules that contain a negative literal and then grounding the remaining (positive) rules)
  - $P^+$  is the smallest possible reduct of P,
  - $T_{P^+}^{\omega}(\emptyset)$  derives all atoms that can be derived by only the remaining purely positive rules,
  - this includes all facts (recall fact rules of the form p(...) :- true.)
  - $\Rightarrow$  these are atoms that hold in *all* models of *P* (facts+positive rules force them).
  - $\Rightarrow$  a safe and very careful underestimate of true atoms.

$$\emptyset \subseteq T^1_{P^{\mathsf{facts}}}(\emptyset) \subseteq T^{\omega}_{P^+}(\emptyset) \subseteq T^{\omega}_{P^{\mathsf{anyl}}}(\emptyset) \subseteq \mathcal{HB}_P$$

640

# Alternating Fixpoint: Analysis

Consider now the program  $P^-$  which is obtained from ground(P) by simply deleting all negative literals from all rules (corresponds to "all negative literals are satisfied"):

- $P^-$  is the reduct wrt. the empty interpretation, the starting point of the whole process,
- $P^-$  it is the biggest possible reduct of P
- $T^{\omega}_{P^-}(\emptyset)$  derives all atoms that can be derived by P if all negative literals are assumed to be satisfied.
- this includes again all facts (recall fact rules of the form p(...) :- true.)
- and everything that could by derived from them under "optimal" conditions
- $\Rightarrow$  an *overestimate* of true atoms.
- $\Rightarrow$  atoms that are not in  $T_{P^{-}}(\emptyset)$  can definitely not be derived by P,
- ⇒ a safe *underestimate of false atoms* (in any stable model/wrt. Closed-World Assumption).
  - Example: Consider  $P = \{p(a), p(b):-not p(a)\}$ . Then,  $P^- = \{p(a), p(b):-true\}$  and  $T^{\omega}_{P^-}(\emptyset) = \{p(a), p(b)\}$ .
  - use this for starting with  $I_0 = \emptyset$  and thus considering  $P^{\emptyset} = P^-$ :

 $\emptyset \subseteq T^1_{P^{\mathsf{facts}}}(\emptyset) \subseteq T^{\omega}_{P^+}(\emptyset) \subseteq T^{\omega}_{P^{anyI}}(\emptyset) \subseteq T^{\omega}_{P^-}(\emptyset) = T^{\omega}_{P^{\emptyset}}(\emptyset) \subseteq \mathcal{HB}_P$ 

# Well-Founded Semantics Computation: Intuitive Analysis

- $\Rightarrow$  coming back to the inductive definition:
  - $I_0 = \emptyset$ ,
  - $I_1 = T^{\omega}_{P^{\emptyset}}(\emptyset)$  is an overestimate of true atoms and an underestimate of false atoms.
  - observation: the larger *I*, the *smaller* the reduct  $P^I$  (delete non-satisfied negative literals), the smaller  $T^{\omega}_{P^I}(\emptyset)$  ("antimonotonic")
  - $P^{I_1}$  is a "small" reduct program,  $T^{\omega}_{P^{I_1}}$  is a "small" interpretation, but  $\supseteq T^{\omega}_{P^+}(\emptyset)$
  - $P^{I_2}$  is a "large" reduct program,  $T^{\omega}_{P^{I_2}}$  is a "large" interpretation, but  $\subseteq T^{\omega}_{P^{\emptyset}}(\emptyset)$



<sup>642</sup> 

Alternating Fixpoint: Analysis

$$I_0 := \emptyset$$
  
$$I_{i+1} := T^{\omega}_{P^{I_i}}(\emptyset)$$

- in each step,  $P^{I_i}$  encodes the knowledge about false atoms from  $I_i$  into P.
- $T^{\omega}_{P^{I_i}}$  runs the resulting positive program under consideration of these false atoms:
- if  $I_i$  is an underestimate of false atoms:
  - only negative literals that are already proven to be true are assumed to be true.
  - $\Rightarrow$  underestimate of the satisfied rule bodies,
  - $\Rightarrow$  underestimate of the true heads.
  - $\Rightarrow I_{i+1} = T^{\omega}_{P^{I_i}}$  is an underestimate of true atoms.
- Analogously, if  $I_i$  is an overestimate of false atoms,  $I_{i+1} = T_{P^{I_i}}^{\omega}$  is an overestimate of true atoms.

## Alternating Fixpoint: Analysis

$$I_0 = \emptyset$$
  
$$I_{i+1} = T_{P^{I_i}}^{\omega}(\emptyset)$$

- $I_0$  is an underestimate of true atoms and an overestimate of false atoms,
- $I_1$  is an overestimate of true atoms and an underestimate of false atoms,
- $I_{2n}$  is an underestimate of true atoms and an overestimate of false atoms,
- $I_{2n+1}$  is an overestimate of true atoms and an underestimate of false atom,
- and with each step, the estimates get better.
- To be proven by interleaved induction:
  - increasing sequence of underestimates:  $I_{2(n+1)} \ge I_{2n}$  (base case obvious:  $I_2 \ge I_0 = \emptyset$ )
  - decreasing sequence of overestimates:  $I_{2n+3} \ge I_{2n+1}$  (first element  $I_1 = T^{\omega}_{P^{\emptyset}}(\emptyset) = T^{\omega}_{P^-}(\emptyset)$  (cf. Slide 641)





#### Alternating Fixpoint: Analysis

#### Lemma 11.1

The mapping  $I \to T_{PI}^{\omega}(\emptyset)$  is antimonotonic: If  $I \leq J$ , then  $T_{PI}^{\omega}(\emptyset) \geq T_{PJ}^{\omega}(\emptyset)$ .

**Proof**  $I \leq J$  means that  $I \subseteq J$ , i.e., in I more atoms evaluate to false. Thus, in  $P_I$  more negative literals are removed (because they are satisfied in I), thus less rules are removed due to remaining negative literals (which are not satisfied). Thus,  $P_I \supseteq P_J$  (as sets of ground rules), thus  $T^{\omega}_{P_I}(\emptyset) \supseteq T^{\omega}_{P_J}(\emptyset)$ .

646

#### Alternating Fixpoint: Analysis

#### Theorem 11.1

With the above definition,  $I_0 \leq I_2 \leq \ldots \leq I_{2n} \leq I_{2n+2} \leq \ldots \leq I_{2n+1} \leq I_{2n-1} \leq \ldots \leq I_1$ .  $\Box$ 

 $\begin{array}{l} \text{Proof Obviously, } I_{0} = \emptyset \leq I_{1} \text{ and } I_{0} \leq I_{2}. \text{ Thus, } I_{2} = T_{P^{I_{1}}}^{\omega}(\emptyset) \leq T_{P^{I_{0}}}^{\omega}(\emptyset) = I_{1}.\\ I_{3} = T_{P^{I_{2}}}^{\omega}(\emptyset) \leq T_{P^{I_{0}}}^{\omega}(\emptyset) = I_{1}.\\ \text{Analogously by induction:}\\ \text{Since } I_{2n-1} \geq I_{2n+1}: \ I_{2n+2} = T_{P^{I_{2n+1}}}^{\omega}(\emptyset) \geq T_{P^{I_{2n-1}}}^{\omega}(\emptyset) = I_{2n}.\\ \text{Since } I_{2n-2} \leq I_{2n}: \ I_{2n+1} = T_{P^{I_{2n}}}^{\omega}(\emptyset) \leq T_{P^{I_{2n-2}}}^{\omega}(\emptyset) = I_{2n-1}.\\ \text{Since } I_{2n+1} \geq I_{2n}: \ I_{2n+2} = T_{P^{I_{2n+1}}}^{\omega}(\emptyset) \leq T_{P^{I_{2n-2}}}^{\omega}(\emptyset) = I_{2n+1}.\\ \text{Since } I_{2n} \leq I_{2n-1}: \ I_{2n+1} = T_{P^{I_{2n}}}^{\omega}(\emptyset) \geq T_{P^{I_{2n-1}}}^{\omega}(\emptyset) = I_{2n}. \end{array}$ 

- The  $I_{2n}$  are a monotonically increasing (and limited) sequence: the underestimates of true atoms.
- The  $I_{2n+1}$  are a monotonically decreasing (and limited) sequence: the overestimates of true atoms.

• 
$$\lim_{n \to \infty} I_{2n} \le \lim_{n \to \infty} I_{2n+1}$$

do the limits coincide? – sometimes yes, but not always!


648



- *I*<sub>3</sub> = {move(a,b), move(b,a), move(b,c), move(c,d), win(c), win(b), win(a)} = *I*<sub>1</sub> win(b) is still there since there is the move to a.
- From then  $(n \ge 2)$  on,  $I_{2n} = I_2$  and  $I_{2n+1} = I_1$ .

# How to interpret this?

- all facts in  $\lim_{n \to \infty} I_{2n}$  have a well-founded derivation "to hold": win(c).
- all facts not in  $\lim_{n \to \infty} I_{2n+1}$  have a well-founded derivation "not to hold":  $\neg$ win(d).
- all others: ?? game: a and b are drawn positions.

What about a logical semantics? - three-valued logic: true/false/undefined.

# EXAMPLE: WIN-MOVE-GAME IN DATALOG

• XSB: use tnot (tabled!) - applies SLG resolution (SLD + memoing/tabling)

```
:- auto_table.
pos(a). pos(b). pos(c). pos(d).
move(a,b). move(b,a). move(b,c). move(c,d).
Win(X) :- move(X,Y), tnot win(Y).
lose(X) :- pos(X), tnot win(X).
% ?- win(X)
```

[Filename: Datalog/winmovesmall.P]

• c is won, d is lost, a and b are undefined (to be interpreted as drawn).

Aside: References

• The win-move game is used in the above-mentioned papers [M. Gelfond, V. Lifschitz, ICLP 1988], [A. Van Gelder, K.A. Ross, J.S. Schlipf, PODS 1988], [A. Van Gelder, PODS 1989].



# 11.3 3-Valued Logic

- same syntax as FOL
- truth values t (true, 1), u (undefined, 0.5), f (false, 0), ordered by t > u > f.
- All three-valued logics coincide in the definition of  $\land$ ,  $\lor$ , and  $\neg$ :

• there is not a single 3-valued logic. There are multiple variants, depending on what should be done with the logic.

#### 3-Valued Logic for Logic Programming Semantics

- · does not require actual reasoning in a 3-valued world,
- · define a model theory for Datalog with negation,
- express partial models:
  - consider Datalog with disjunction in the head (or similar situations e.g. in Description Logics/OWL):

Consider an axiom  $\forall X : person(X) \rightarrow (male(X) \lor female(X)).$ 

Consider an interpretation  $\mathcal{I}$  where there is an individual a s.t.  $\mathcal{I} \models person(a)$ . From  $\mathcal{I} \models \forall X : person(X) \rightarrow (male(X) \lor female(X))$ .

the intended semantics of  $\models$  and  $\rightarrow$  (both must still be defined!) should imply that  $\mathcal{I} \models \mathsf{male}(a) \lor \mathsf{female}(a)$ .

Since it is not known whether *a* is male or female, the model theory for partial models with negation in the head should allow that neither male(a) nor female(a) belong to  $\mathcal{I}$ .

- this chapter: allow to define and compute  $T_P(I)$  for rules with negation in the body:
  - evaluate conjunctive bodies with negation,
  - $T_P$  for such rules: if the truth value of the body is u, that of the head should also be u.
  - an appropriate notion for  $I \models P$  for partial interpretations wrt. such programs.

# 3-Valued Logic: Implication

For implication, there are different definitions (here, only two are listed):

- 1. Logic K<sub>3</sub>, Stephen Kleene (1938):  $A \rightarrow B = \neg A \lor B = \max(1 - A, B)$ follows the definition of  $\rightarrow$  as a derived operator from boolean logic. u t
  - Fits with intuitive "if the truth value of the body is unknown and the truth value of the head is unknown, then the truth value of  $A \rightarrow B$  is also unknown".
  - Does not fit with the intention to handle *I* ⊨ *head* ← *body* where the truth value of the body (and that of the head) is *u*.
- 2. based on the ordering of the domain: t > u > f:  $A \rightarrow B = (A \le B)$ 
  - the truth value of  $A \rightarrow B$  is always t or f,
  - For a rule  $head \leftarrow body$ , if I(body) = u and I(head) = u, then  $I(head \leftarrow body) = t$ .

$$\begin{array}{c|cccc} B:head\\ f & u & t \end{array}$$
 
$$\begin{array}{c|ccccc} hpoq\\ hpoq\\ u & f & t & t\\ F & t & f & f & t \end{array}$$

 $f \quad u \quad t$ 

t t

f

<u>u</u> u t

<u>u</u> t

 $\Rightarrow$  use the second alternative.

654

# **3-VALUED LOGIC: NOTATION AND MINIMAL MODELS**

Extend and adapt FOL notation:

- 3-valued Herbrand interpretations are given as tuples I = (T, F) where T is the set of true atoms and F is the set of false atoms. All other atoms are undefined.
- *I*<sub>1</sub> ≤ *I*<sub>2</sub> is defined wrt. the amount of information:
   with partial order *u* ≺ *t* and *u* ≺ *f*
  - $I_1 \leq I_2$  if for all ground atoms a ,  $I_1(a) \preceq I_2(a)$ ,
  - or equivalently  $(T_1, F_1) \leq (T_2, F_2) \Leftrightarrow T_1 \subseteq T_2$  and  $F_1 \subseteq F_2$ .
- The minimal interpretation is thus formally correctly written as  $(\emptyset, \emptyset)$ .
- instead of *I* ⊨ φ or *I* ⊨<sub>β</sub> φ (which can only express true/false), write *I*(φ) = v or val<sub>*I*,β</sub>(φ) = v for v ∈ {t, u, f}. Convention: write *I* ⊨ φ ("*I* is a model of φ") in 3-valued context if *I*(φ) = t. (⊨ will only be applied to programs and rules, the semantics of → has been defined above to result in t or f.)

# 11.4 3-Valued Well-Founded Model

Given a program P, define a certain 3-valued Herbrand interpretation I = (T, F) as follows;

# **Definition 11.5**

For a Datalog<sup>¬</sup> program P with  $I_0 = \emptyset, I_1, \dots, I_{2n}, I_{2n+1}, \dots$  the Alternating Fixpoint Computation, let  $\mathcal{W}_P := (\{a | a \in \lim_{n \to \infty} I_{2n}\}, \{a | a \in \mathcal{B}_P, a \notin \lim_{n \to \infty} I_{2n+1}\})$ .

- "true": all facts that are in the final underestimate of true atoms;
- "false": all facts that are outside of the final overestimate of true atoms they are definitely false.

It will be proven later that  $W_P$  is the well-founded model of P (cf. Definition 11.3).

656

#### Example

Consider again the simple win-move game from Slide 649.

The corresponding program is P =

```
pos(a). pos(b). pos(c). pos(d).
```

```
move(a,b). move(b,a). move(b,c). move(c,d).
```

win(X) :- move(X,Y), not win(Y).

lose(X) :- pos(X), not win(X).

[Filename: Datalog/winmove-small.s]

With the sequence  $((I_k))$  as given on Slide 649, the alternating fixpoint computation stops at  $I_3 = I_1$  (EDB shown in gray):

 $\mathcal{W}(P) = (\{ pos(a), pos(b), pos(c), pos(d),$ 

move(a,b), move(b,a), move(b,c), move(c,d), win(c), lose(d)}

{ move(a,a), move(a,c), move(a,d), move(b,a), move(b,b), move(b,d), move(c,a), move(c,b), move(c,c), move(d,a), move(d,a), move(d,b), move(d,c), move(d,d), win(d), lose(c)} )

undefined: win(a), win(b), lose(a), lose(b)

(usually one omits the EDB predicates when listing well-founded or stable models).

# 3-VALUED $\mathbf{T}_P$ -OPERATOR

Definition 10.2 carries over to 3-valued interpretations as follows:

#### Definition 11.6 ( $3T_P$ -Operator)

For a ground Datalog<sup>¬</sup> program  $P_g$  (which might contain the boolean atom undef in the body) and a 3-valued interpretation I = (T, F), for each ground atom a,

 $3T_{P_a}(I)(a) := \max(\{I(body) : a \leftarrow body \in P_q\})$ 

For a non-ground Datalog<sup>¬</sup> program *P* and a 3-valued interpretation I = (T, F),  $3T_P(I) := 3T_{P_g}(I)$  where  $P_g$  is the grounding of *P* wrt. the Herbrand Universe of *P* (i.e., the set of all possible ground instances of the rules of *P*).

$$\begin{array}{rcl} 3T^0_P(I) & := & I \\ 3T^1_P(I) & := & 3T_P(I) \\ 3T^{n+1}_P(I) & := & 3T_P(3T^n_P(I)) \\ 3T^{\omega}_P(I) & := & \bigcup_{n \in \mathbb{N}} 3T^n_P(I) \\ & 3T^{\omega}_P & := & 3T^{\omega}_P(\emptyset, \emptyset). \end{array}$$

658

# **3-VALUED REDUCT**

Definition 11.1 (Slide 631) carries over to 3-valued interpretations as follows:

#### Definition 11.7 (3-Valued Reduct)

For a Datalog<sup>¬</sup> program P, and a 3-valued interpretation I = (T, F), the reduct  $P^I$  of P wrt. I is obtained as follows:

- let  $P_g$  denote the grounding of P,
- delete from  $P_g$  all rules that contain a negative literal  $\neg a$  in the body such that I(a) = t,
- replace all negative literals ¬a in the remaining rules s.t. I(a) = u by the boolean atom undef (since undef is neither in T nor in F it will be evaluated as I(undef) = u),
- delete all remaining negative literals in the bodies of the remaining rules.

#### Properties of $P^I$

- $P^I$  is a ground positive program.
- If *I* is a model of *P*, then for each ground atom *a*,  $(3T^{\omega}_{P^{I}}(\emptyset))(a) \leq I(a)$ .

# **3-STABLE MODELS**

# **Definition 11.8**

A 3-valued interpretation I = (T, F) is a 3-stable model of a Datalog<sup>¬</sup> program P, if

 $3T_{P^{I}}^{\omega}(\emptyset, \emptyset) = I.$ 

For returning also partial models, invoke smodels with --partial.

- output p(a) means that p(a) can be derived to be true
- output p'(a) means that  $val(p(a) \ge u$  is at least undefined (p(a) might also be listed to be true)
- this avoids to have to list all possible ground instantiations of atoms that are false.

(see next slide)

660

# Example/Syntax: Partial Stable Model in smodels

#### Example 11.1

p(a) :- not p(a).

[Filename: Datalog/pnotp.s]

... has only one partial stable model: p(a) is undefined:

```
lparse -n 0 --partial pnotp.s|smodels
smodels version 2.34. Reading...done
Answer: 1
Stable Model: p'(a)
```

Interpretation of the result  $M = \{p'(a)\}$  (smodels Section 4.8.2):

- for every ground atom p(...), an atom p'(...) is added to the internal program, which means "p(...) is potentially true"
- if both  $p(\ldots)$  and  $p'(\ldots)$  are in M, then  $val_M(p(\ldots)) = t$ ,
- if  $p'(\ldots) \in M$  and  $p(\ldots) \notin M$ , then  $val_M(p(\ldots)) = u$ ,
- otherwise  $val_M(p(\ldots)) = f$ .

#### Example

## Example 11.2

Consider once more the program from Slide 634:

q(a) :- not p(a). p(a) :- not q(a).

[Filename: Datalog/porq.s]

Exercise: give the Alternating Fixpoint Computation for P.

- $S := (\emptyset, \emptyset)$ , *i.e.*, S(p(a)) = S(q(a)) = u, *is a 3-stable model. It is the minimal 3-stable model.*
- On Slide 634, {p(a)} and {q(a)} have been identified as total stable models of P.
   Note: as partial models, these are written as (T, F)-pairs as ({p(a)}, {q(a)}) and ({q(a)}, {p(a)})

# Example 11.3

Consider winmove.p with --partial.

- here, the unique partial stable model (= the well-founded model) is the "intended" one with drawn positions.
- the total stable models arbitrarily "fix" some drawn positions to be won/lost (in an admissible way wrt. the program).

662

# WELL-FOUNDED MODEL

Recall Definition 11.3 (638):

For a Datalog<sup>¬</sup> program P, the (in general three-valued) well-founded model of P is the (unique) minimal 3-stable model of P.

#### Theorem 11.2

 $\mathcal{W}_P$  (as defined on Slide 656) is the well-founded model of P.

Proof:

- Show that  $W_P$  is 3-stable [Abiteboul, Hull, Vianu: Foundations of Databases, Thm. 15.3.9]
- minimality and uniqueness follow from Lemma 11.2:

#### Lemma 11.2

For a Datalog<sup>¬</sup> program P,  $W_P = (T, F)$  is the intersection of all 3-stable models of P, i.e., for every 3-stable model (T', F'),  $T' \supseteq T$  and  $F' \supseteq F$ .

Proof: minimality of T wrt. all models and minimality of F wrt. all stable models follows from the properties proven for the AFP computation.

Comments: Well-Founded Model

- The AFP gives a (polynomial!) computation for the non-constructive definition of "well-founded model".
- all stable models extend the well-founded model
   ⇒ computation/guessing can be based on the well-founded model.
- starting the Alternating Fixpoint Computation with the contents of the EDB relations as initial interpretation  $J_0$  leads to the same final result (but the intermediate  $J_i$  are different and  $J_0$  serves as an underestimate).

664

# Recall: Non-Monotonicity of Closed-World-Assumption

"Negation by default" is non-monotonous:

Consider a program P and its well-founded model W(P) = (T, F):

• recall that any program (we have only positive atoms in the head) cannot imply that any atom *must be* false in all models

 $\Rightarrow$  any positive fact can be added to a Datalog/Datalog^ program without being inconsistent.

- there are non-stable models  $\mathcal{M} = (T', F')$  of P where  $T' T \neq \emptyset$  (containing atoms that are not supported by P), and for these, often also  $F F' \neq \emptyset$ 
  - e.g. add an edge to the win-move game, and some other positions are won, but some that were won before are now lost, or
  - e.g. just fix that a certain (drawn or even lost) position is won.
  - $F F' \neq \emptyset \Rightarrow$  Things that have been concluded before to hold do now turn out not to hold; "Belief Revision".
- $\mathcal{M}$  is then a 3-stable model of a (more or less slightly) different program  $P' \supseteq P$ .

(e.g.,  $P' = P \cup \{move(x,y)\} \text{ or } P' = P \cup \{win(x)\})$ 

 $\Rightarrow$  corresponds to "learning" about a new fact,

 $\Rightarrow$  requires to recompute the whole well-founded model from scratch.

## Exercise: Well-Founded Model

- show that for every positive Datalog program *P*, the well-founded model is *total* (i.e., all ground atoms are either true or false).
- show that for every stratifiable Datalog<sup>¬</sup> program P, the well-founded model is *total*.

Exercise: Well-Founded Model

- Are there non-stratifiable Datalog<sup>¬</sup> programs that have a total well-founded model (i.e., no atoms undefined)?
- Are there (non-ground) non-stratifiable Datalog<sup>¬</sup> programs that have a total well-founded model for *all* EDB instances?

666

#### Well-founded Semantics: Literature

- · definition of reduct and stable model taken from documentation of smodels,
- alternating fixpoint taken from ??TO BE EXTENDED??
- further reading: [Abiteboul, Hull, Vianu: Foundations of Databases]
- Original Paper: Allen Van Gelder, Kenneth A. Ross, John S. Schlipf: Unfounded Sets and Well-Founded Semantics for General Logic Programs. PODS 1988: 221-230
- Long version: Allen Van Gelder, Kenneth A. Ross, John S. Schlipf: The Well-Founded Semantics for General Logic Programs. J. ACM 38(3): 620-650 (1991)
- Alternating Fixpoint: Allen Van Gelder: The Alternating Fixpoint of Logic Programs with Negation. PODS 1989: 1-10
- online literature database (started with database + logic programming, now for everything in CS): http://dblp.uni-trier.de/ (from university computers, access to most pdfs is allowed)

# RESTRICTIONS OF THE DATALOG/MINIMAL/WELL-FOUNDED MODEL SEMANTICS

Given a Datalog/Datalog<sup>¬</sup> program P, the minimal model, well-founded model, and the AFP procedure cannot decide the following:

- for a given general FOL formula  $\phi$ , does  $\phi$  hold in *all* models of *P*?
- if p(c<sub>1</sub>,...,c<sub>n</sub>) can not be confirmed by the minimal, stratified, or well-founded model, this does *not* mean, that there is no model of P where p(c<sub>1</sub>,...,c<sub>n</sub>) holds.
   Even more, any positive fact can be added to a Datalog/Datalog<sup>¬</sup> program without being inconsistent.

#### Closed-World-Assumption (CWA)

- For all facts that are not given in the database and that are not derivable, it is assumed that they do not hold (more explicitly: that their negation holds).
- CWA not appropriate in the Web: for things that I do not find in the Web, simply nothing is said.

[Example: travel planning]

668

# THE LIMITS – NO REAL DISJUNCTION

	$7_n(X)$
:- auto_table.	p(x)
p(a) := tnot p(b).	X = b undefined
r(h) $t$ that $r(h)$	X = a undefined
p(b) := chot p(a).	?- q(X).
q(c) :- p(X).	X = c undefined
Filename: Datalog/pg.P]	
	X = c undefined

- "q(c) undefined" is computed twice by SLG resolution, i.e. two proof paths exist.
- $W(P) = (\emptyset, \{q(a), q(b), p(c)\})$ , the "interesting" ground atoms  $\{p(a), p(b), q(c)\}$  are undefined. The model theories of the minimal model and well-founded model define truth/entailment only for ground atoms.

 $\{p(b), p(a)\} \quad \{\neg p(X), q(c)\}$ 

 $\{p(b)\}$ 

 $\{\neg p(X_1)\}$ 

 $\{\neg q(c)\}$ 

 $\{\neg p(X_2)\}$ 

- *P* as a FOL formula:  $(p(b) \lor p(a)) \land \forall x : p(x) \to q(c) \models_{\mathsf{FOL}} q(c)$
- (general) resolution proof: clauses  $\{p(a), p(b)\}$  (which is the clause corresponding to both the two first rules) and  $\{\neg p(X), q(c)\}$  together with query/goal clause  $\neg q(c)$  allow to derive  $\Box$ :



# THE LIMITS – NO REAL DISJUNCTION

The same program interpreted by stable models:

```
thing(a). thing(b). thing(c).
p(a) :- not p(b).
p(b) :- not p(a).
q(c) :- thing(X), p(X).
[Filename: Datalog/pg.s]
```

```
lparse -n 0 --partial pq.s|smodels
models version 2.34. Reading...done
Answer: 1
Stable Model: p(a) q(c)
Answer: 2
Stable Model: p(b) q(c)
## Answer: 3
## Stable Model: p'(a) p'(b) q'(c)
False
```

- two total stable models:
  - "either p(a) or p(b) hold"
  - "q(c) holds in any case"
- the user can interpret the result as a 3-valued interpretation *I* where val<sub>I</sub>(p(a)) = val<sub>I</sub>(p(b)) = u and val<sub>I</sub>(q(c)) = t.

*I* is a model of *P* (i.e.,  $3T_{P_I}^{\omega}(\emptyset) \leq I$ ), but *I* is *not* a *stable* model of *P* (i.e.,  $3T_{P_I}^{\omega}(\emptyset) \neq I$ )!

670

# Chapter 12 Stable Models/Answer Set Programming

- ASP developed in the late 1990s.
- Introduction to ASP: Answer Set Programming: A Primer. T. Eiter, G. Ianni, T. Krennwallner. In "Reasoning Web. Semantic Technologies for Information Systems", Springer LNCS 5689, 2009.
- XSB: XASP package: embeds stable models into XSB PROLOG programming. Not suitable for this lecture.
- smodels+lparse: http://www.tcs.hut.fi/Software/smodels/

# SITUATION

Usually a program has several stable models (otherwise, the well-founded model is sufficient!)

- either one total well-founded model or one partial well-founded model,
- and either zero or more total stable models, or zero or more partial stable models
- if the well-founded model is total, then "everything is clear" and it is the only stable model.
- if the well-founded model W(P) = (T, F) is partial, its T and F are "guaranteed". Stable models deal with the atoms that are undefined in the well-founded model (= the intersection of all stable models).

According to Lemma 11.2, each stable model is an extension of  $W_P = (T, F)$ , i.e., for every 3-stable model (T', F'),  $T' \supseteq T$  and  $F' \supseteq F$ .

672

#### Example: Win-Move Game

Consider again the small win-move game from Slides 649 and 657:

```
pos(a). pos(b). pos(c). pos(d).
move(a,b). move(b,a). move(b,c). move(c,d).
win(X) :- move(X,Y), not win(Y).
lose(X) :- pos(X), not win(X).
```

[Filename: Datalog/winmove-small.s]

The well-founded model has been derived on Slide 657:

 $W(P) = (\{win(c), lose(d)\}, \{lose(c), win(d)\})$  is partial, since win(a)/lose(a) and win(b)/lose(b) are undefined.

(Total) Stable Models:

 $\mathcal{S}_1(P) = (\{\text{win}(a), \text{win}(c), \text{lose}(b), \text{lose}(d)\}, \{\text{win}(b), \text{win}(d), \text{lose}(a), \text{lose}(c)\}) \text{ and } \mathcal{S}_1(P) = (\{\text{win}(b), \text{win}(c), \text{lose}(a), \text{lose}(d)\}, \{\text{win}(a), \text{win}(d), \text{lose}(b), \text{lose}(c)\}).$ 

Here,  $W(P) = (\{win(c)\}, \{win(d)\})$  provides the "intended" application-specific interpretation of the result: *c* is won, *d* is lost, *a* and *b* are drawn.

From the stable models one can only conclude that it is possible to "fix" a to be won, and then b would also be lost (or vice versa).

# Example: Choice between Alternatives

Consider again the program P = "porq" from Slide 634 and 662:

q(a) :- not p(a). p(a) :- not q(a).

[Filename: Datalog/porq.s]

Three stable models: lparse -n 0 --partial pq.s|smodels

- $\mathcal{W}(\mathcal{P}) = (\emptyset, \emptyset)$  is the well-founded model, and a partial stable model,
- $\{p(a)\}$  whose 3-valued representation is  $(\{p(a)\}, \{q(a)\})$  is a total stable model, and
- $\{q(a)\}$  whose 3-valued representation is  $(\{q(a)\}, \{p(a)\})$  is also a total stable model.

Depending on the application

- the well-founded model tells that nothing is known.
- the two stable models tell what are the two possibilities that have to be considered (in a more complex scenario, this would already exclude some alternatives)
- underspecification: the two stable models tell that the user is allowed to state additional facts (according to his preferences).

674

# 12.1 Answer Set Programming

Different idea than

- Stratified Datalog<sup>¬</sup>: Query Answering
- Prolog: "Declarative" Prolog Programming the "search" is encoded into the SLD-resolution-tree.
- Rules as derivation: "if body holds, then derive head".

Answer Set Programming: Specify a problem declaratively and leave the reasoning to the ASP solver (from [EIK09]):

- Rules as assertions: "if body holds, then head holds".
- Allow disjunction and negation in the head.
- Possibility of modeling constraints;
- Reasoning with incomplete information; and
- Possibility of modeling preferences and priority.
- Spatial and temporal reasoning (here, the notorious Frame Problem is challenging).

# SMODELS + LPARSE

- call lparse -n 0 -d none [-partial] filename | smodels
- smodels requires "..." for strings,
- smodels does not accept the "\_" don't-care underscore.
   Use X1, X2 etc. and constrain it by a domain predicate (see next slide).
- smodels does not accept decimal numbers only (positive and negative) integers
  - decimals would not allow a grounding of the program. (integers are only little better ...)
  - $\Rightarrow$  thus, mondial.P cannot be used.

676

#### Grounding the Program in smodels: Domain Predicates

(note: important for writing programs)

- Computation of stable models is based on grounding the program (cf. generating the reduct of a program on Slide 631)
- For grounding the rules (i.e., generate all relevant ground instances of each rule), smodels internally looks for *"domain predicates"* whose extension can be precomputed by a simple Datalog<sup>¬</sup> subprogram of *P* without extended rules (cf. smodels Manual Sections 4.4.2 and 4.4.3):
  - union, intersection, join, set difference (with one negative literal!)
- every rule must contain at least one domain predicate atom.
- the ground instances of the domain predicates are precomputed and used for grounding.
   (cf. the example for the reduct of the win-move game generate only ground instances of win(X) :- move(X,Y), not win(Y) s.t. move(a,b) is in the EDB).
- If Iparse/smodels complains about

```
<line_nr>: weakly restricted rule: <....>.
    weakly restricted variables: <var>
```

introduce a domain predicate and use it in the rule (in the following: thing(X) with appropriate definition of thing).

# DISJUNCTION REVISITED: STABLE MODELS

```
mainDish(X) :- meal (X,Y).
            :- meal (X,Y).
drink(Y)
vegetarian(X) :- mainDish(X), not nonvegetarian(X).
                                                          %% xor(veg, nonveg)
nonvegetarian(X) :- mainDish(X), not vegetarian(X).
porc(X) :- mainDish(X), nonvegetarian(X), not beef(X), not fish(X). %% xor (porc,beef,fish)
beef(X) :- mainDish(X), nonvegetarian(X), not porc(X), not fish(X).
fish(X) :- mainDish(X), nonvegetarian(X), not porc(X), not beef(X).
whitewine(Y) :- meal(X,Y), fish(X).
whitewine(Y) :- meal(X,Y), porc(X).
redwine(Y) :- meal(X,Y), beef(X).
wine(Y) :- drink(Y), redwine(Y).
                                                   %% wine = redwine u whitewine
wine(Y) :- drink(Y), whitewine(Y).
alcoholic(Y) :- drink(Y), wine(Y).
tomatojuice(Y) :- meal(X,Y), vegetarian(X).
meal(a,b).
nonvegetarian(a).
                                                [Filename: Datalog/meals.s and meals.P]

    consider the question: is b alcoholic?
```

• meal/2, mainDish/1 and drink/1 act as domain predicates.

678

#### Example (Cont'd)

- Three (total) stable models: *a* is either porc, beef or fish. In either case, white/red wine is served with it, thus alcoholic(*b*) holds.
- Exercise: give the AFP computation.
- well-founded model:

- 
$$val_{\mathcal{W}(P)}(\mathsf{fish}(b)) = val_{\mathcal{W}(P)}(\mathsf{porc}(b)) = val_{\mathcal{W}(P)}(\mathsf{beef}(b)) = u$$
,

- $val_{\mathcal{W}(P)}(\mathsf{alcoholic}(b)) = u$
- note: XSB yields a=fish, not computing the well-founded model.

(cf. also formal example from Slide 669)

## Logical Rules: Local and Global Semantics

- Wrt. logical rules, the concept of disjunction does not exist: there are several *isolated* rules that *derive* wine(X) (in Description Logics, wine would be a union class Wine = RedWine ⊔ WhiteWine).
- The *global* semantics of the well-founded model and of stable models is always based on the *local* semantics of rules.
- any application-specific semantics (= interpretation of the model) has to be defined outside of the LP framework.
- Consider I agreeing with the well-founded model up to val<sub>I</sub>(fish(a)) = val<sub>I</sub>(porc(a)) = val<sub>I</sub>(beef(a)) = val<sub>I</sub>(whitewine(b)) = val<sub>I</sub>(redwine(b)) = u, but then setting val<sub>I</sub>(wine(b)) = val<sub>I</sub>(alcoholic)(b) = t: I \models P, but I is not stable!
- Recall Lemma 11.2: " $W_P = (T, F)$  is the intersection of all 3-stable models of P". Above example: This may be smaller than the intersection of all 2-stable models!

680

# **APPLICATION-SPECIFIC INTERPRETATION OF LP NOTIONS**

- consider all total stable models and
  - "cautious reasoning": take facts that are true/false in all of them. (results are models of P, but not necessary stable ones) (meals example: alcoholic(b))
  - "credulous reasoning": take facts that are true/false in one of them (they are possible - somebody reporting them may tell the truth) (meals example: whitewine(b), redwine(b), but tomatojuice(b) is definitely false)
- preference by the user
  - interpreted as underspecification:
  - choose a "desired" fact a that is undefined and consider only models that satisfy it,
  - add *a* to the program and run again
    - without negation in the head/denials (cf. Slide 686):
       one or more new 3-stable models exist
- systematic weight (lparse/smodels support weighted clauses)

# **DISJUNCTION IN RULE HEADS**

(this syntax is buggy in smodels; see next page for (more expressive) alternative)

• rules of the form

```
A_1|\ldots|A_n := B_1,\ldots,B_m
```

(recall: conjunction is expressed by n rules with the same body)

- · stable models in presence of disjunction are not necessarily minimal
- · check of minimality is again NP-complete
- · smodels yields all stable models
- smodels, Section 4.8.1
- invoke with --dlp
- smodels returns "Error in input" (lparse without | smodels accepts it)

```
a | b :- c.
c :- not d.
```

(Total) stable models:

```
d :- not c.
```

[Filename: Datalog/disj.s]

682

•  $\{d\}, \{c, a\}, \{c, b\}, \{c, a, b\}.$ 

# SMODELS: DISJUNCTION IN THE HEAD VIA CHOICE RULES

- extended rule head:  $k \{a_1, \ldots, a_n\} m := body$
- *k* to *m* atoms of the head must be true if *body* is true.

```
mainDish(X) :- meal (X,Y).
drink(Y) :- meal (X,Y).
1{vegetarian(X),nonvegetarian(X)}1 :- meal(X,Y).
1{porc(X), beef(X), fish(X)}1 :- mainDish(X), nonvegetarian(X).
whitewine(Y) :- meal(X,Y), fish(X).
whitewine(Y) :- meal(X,Y), porc(X).
redwine(Y) :- meal(X,Y), beef(X).
wine(Y) :- drink(Y), redwine(Y).
wine(Y) :- drink(Y), whitewine(Y).
alcoholic(Y) :- drink(Y), wine(Y).
tomatojuice(Y) :- meal(X,Y), vegetarian(X).
meal(a,b).
nonvegetarian(a).
[Filename: Datalog/meals-disj.s]
```

#### Stable Models – Example

Consider the following program (cf. Slide 635):

1{p(a), q(a)}1. p(a).

[Filename: Datalog/pporq-choice.s]

- The program has only one *total* stable model: {*p*(*a*)}. (which corresponds to the 3-valued ({*p*(*a*)}, {*q*(*a*)}.)
- The 3-valued interpretation ({p(a)}, ∅) (Stable Model: p(a) p'(a) q'(a)) is also considered a (partial) stable model (!) [Bug or not bug?]

684

# **EXPLICIT NEGATION**

- well-founded and stable semantics are still based on default negation. non-monotonic: adding some positive atom is always allowed and may make before conclusions invalid (and add others).
- explicit negation:
  - $\neg p(a_1, \ldots, a_n)$  as negative facts,
  - $\neg p(X_1, \ldots, X_n)$  :- ... in rule heads.
- Explicit negation is monotonic. "Additional" positive information wrt. such atoms would make the program inconsistent.
- applications:
  - in diagnosis systems to explicitly derive negative knowledge,
  - for expressing integrity constraints.
- XSB: WSFX package, XSB Manual Part II.
- smodels: only integrity constraints via *denials* (see next slide).

# **EXPLICIT NEGATION VIA DENIALS**

A *denial* is a constraint that forbids certain database states (DB: integrity constraints can be formulated as denials)

· Consider a "rule"

:- p.

Its semantics is "if p is true, then the empty clause is true", i.e., then, "false" is true.

• means "p must not be true".

```
:- p(a).
q(a) :- not p(a).
p(a) :- not q(a).
```

[Filename: Datalog/porq-denial.s]

- the program has only one stable model,  $\{q(a)\}$ . It is total.
- note: XSB does not care for such rules (they would be useless there since XSB computes only one model).

686

#### Example: Italians Revisited

The italian-vs-english ontology from Slide 546 can be specified via choice rules (and implicit) denials (line 1: either italian or english allowed):

<pre>0{ italian(X), english(X) }1 :- person(X),</pre>	thing(X)
<pre>person(X) :- italian(X),</pre>	thing(X)
<pre>person(X) :- english(X),</pre>	thing(X)
<pre>1{ lazy(X), latinlover(X) }1 :- italian(X),</pre>	thing(X)
<pre>italian(X) :- lazy(X),</pre>	thing(X)
<pre>italian(X) :- latinlover(X),</pre>	thing(X)
<pre>0{ gentleman(X), hooligan(X) }2 :- english(X),</pre>	thing(X)
<pre>english(X) :- gentleman(X),</pre>	thing(X)
<pre>english(X) :- hooligan(X),</pre>	thing(X)
<pre>gentleman(X) :- latinlover(X),</pre>	thing(X)
italian(e).	
thing(e).	

[Filename: Datalog/italians-english.s]

- do not query for ?- lazy(e) , but inspect the stable model(s).
- there is a single (total) stable model where lazy(e) holds.

# SMODELS: MULTIPLE MODELS

• Motivated by the "Ascending, Descending" graphics by M.C.Escher http://en.wikipedia.org/wiki/Ascending\_and\_Descending

```
corner(1..4). % set of corners
higher(1,2).
1{ higher(1,2), higher(1,3), higher(1,4) }1.
1{ higher(2,1), higher(2,3), higher(2,4) }1.
1{ higher(2,1), higher(2,3), higher(2,4) }1.
1{ higher(3,1), higher(3,2), higher(3,4) }1.
1{ higher(4,1), higher(4,2), higher(4,3) }1.
:- higher(4,1), higher(4,2), higher(4,3) }1.
:- higher(X,X), corner(X). % irreflexive
:- higher(X,X), corner(X). corner(Y). % asymmetric
% inverse functional
:- higher(Y,X), higher(Y,X), corner(X), corner(Y), corner(Z), Y != Z.
% :- higher(X,Y), higher(X,Z), corner(X), corner(Y), corner(Z), Y != Z.
```

• two possibilities = two models

• recall Semantic Web: only answers what holds in all models

688

# SMODELS: PLANNING

Give a specification P of the workflow including constraints:

• if at least one total stable model *S* exists, the specification including the constraints is satisfiable.

 ${\boldsymbol{S}}$  describes the plan that must be followed.

- if several total stable model exist, each one represents a possible execution. Different plans mean that choicepoints during the execution exist.
  - they can be decided a priori: add the intended atoms to P:
    - \* there is at least one stable model, but maybe still several ones.
  - ... or decide them during execution of the workflow (e.g. to be able to react upon external influences).

#### Planning Example: The Farmer's Puzzle

A farmer, travelling to the market with his dog, a goat, and a cabbage. He has to cross a small river, where a boat can be used.

- When using the boat, he can transport only one item.
- He can cross the river as often as he wants.
- When the dog stays on the same side as the goat, and the farmer is not there, the dog will kill and eat the goat.
- When the goat stays on the same side as the cabbage, and the farmer is not there, the goat will eat the cabbage.

Is it possible for the farmer to bring all items to the other side? If yes, how?

Example (Cont'd)	
state(18). % estimate the number of nec	cessary states.
<pre>side(l). side(r).</pre>	
is(farmer,1,1). % farmer is on the left s	side in state 1 (with all items)
<pre>thing(cabbage). thing(goat). thing(dog).</pre>	
is(X,1,1) :- thing(X).	
<pre>otherside(X,Y) :- side(X), side(Y), X != Y.</pre>	
O{ transport(cabbage,N), transport(goat,N), t	ransport(dog,N) }1 :- state(N), not finished(1
:- transport(X,N), is(farmer,S1,N), is(X,S2,N	I), S1 != S2,
<pre>thing(X), state(N), side(S1), side(S2).</pre>	
<pre>is(X,S2,M) :- is(X,S1,N), thing(X), otherside</pre>	P(S1,S2), transport(X,N), M = N+1, not finished
<pre>state(N), side(S1).</pre>	
<pre>is(X,S1,M) :- is(X,S1,N), thing(X), not trans</pre>	<pre>sport(X,N), M = N+1, not finished(N),</pre>
<pre>state(N), side(S1). %% the "frame axid</pre>	om"
<pre>is(farmer,S2,M) :- is(farmer,S1,N), otherside</pre>	e(S1,S2), M = N+1, not finished(N),
<pre>state(N), side(S1).</pre>	
:- is(cabbage,S,N), is(goat,S,N), not is(farm	mer,S,N), state(N), side(S).
:- is(goat,S,N), is(dog,S,N), not is(farmer,S	S,N), state(N), side(S).
<pre>finished(N) :- is(cabbage,r,N), is(goat,r,N),</pre>	is(dog,r,N), state(N).
<pre>finished(M) :- finished(N), M = N+1,</pre>	<pre>state(N).</pre>
:- not finished(8).	[Filename: Datalog/farmer.s]

## Example (Cont'd)

Two stable models:

- first carry the goat to the other side (r) (it must not be left with the dog or with the cabbage).
- 2. go back,
- 3. bring either the dog or the cabbage to the other side,
- 4. go back with the goat,
- 5. bring the cabbage or the dog (one is still there) to the other side,
- 6. go back,
- 7. take the goat and bring it to the right bank again,
- 8. continue traveling to the market.
- $\Rightarrow$  Step 3 is a choicepoint.

SMODELS: SUDOKU SOLVER					
<pre>% example sudoku content from (german) wikipedia p(2,9,3). p(4,8,1). p(5,8,9). p(6,8,5). p(3,7,8). p(8,7,6). p(1,6,8). p(5,6,6). p(1,5,4). p(4,5,8). p(9,5,1). p(5,4,2). p(2,3,6). p(7,3,2). p(8,3,8). p(4,2,4). p(5,2,1). p(6,2,9). p(9,2,5). p(8,1,7).</pre>					
<pre>% general sudoku rules (x = cols, y = rows) col(19). row(19). num(19). % samesquare expresses the 3x3 subsquares: samesq(1,2). samesq(1,3). samesq(2,3). samesq(4,5). samesq(4,6). samesq(5,6). samesq(7,8). samesq(7,9). samesq(8,9). samesq(B,A) :- samesq(A,B), num(A), num(B).</pre> [Filename: Datalog/sudoku.s]					
<pre>1{p(X,Y,1),p(X,Y,2),p(X,Y,3),p(X,Y,4),p(X,Y,5),p(X,Y,6),p(X,Y,7),p(X,Y,8),p(X,Y,9)}1 :- col(Y), row(X). :- p(X,Y1,N), p(X,Y2,N), col(X), row(Y1), row(Y2), num(N), Y1!=Y2. :- p(X1,Y,N), p(X2,Y,N), col(X1), col(X2), row(Y), num(N), X1!=X2. :- p(X1,Y1,N), p(X2,Y2,N), col(X1), col(X2), row(Y1), row(Y2), num(N), X1!=X2.</pre>					
samesq(X1,X2), samesq(Y1,Y2), X1 + 10 * Y1 != X2 + 10 * Y2.					

#### Aside: Another Sudoku

 for most strategy-based solvers it is "unsolvable", requires "trial and error" (which is actually backtracking)





# COMMENTS AND LINKS ON ASP AND SUDOKU SOLVING

- None of the rules is "constructive" in the sense of "position (x, y) must be  $n_1$  if ...", or "position (x, y) must be  $n_1$  of  $n_2$  if ..."
- the description strongly relies on the denials and the solver must find out what is not forbidden.
- Sudoku is a typical *Constraint Satisfaction Problem*; ASP is a certain form of Constraint Solving.
  - See e.g. https://en.wikipedia.org/wiki/Constraint\_satisfaction\_problem.
- Sudoku solved by an explicit Constraint Propagation algorithm in python: http://norvig.com/sudoku.html by Peter Norvig, a well-known Al scientist.

# SMODELS: FURTHER EXAMPLES AND PUZZLES

- the smodels documentation (Section 6.1) contains
  - the Graph 3-Coloring problem (sudoku can also be encoded as graph coloring),
  - some logical puzzles.
- Exercise: encode the Fish Puzzle (see Web page) in ASP and let smodels solve it.

696

# "Prove" $P \models \varphi$

- Stable models:
  - definition (cf. Slide 659) is based on grounding P with the active domain.
  - $\Rightarrow$  Problem solving for concrete cases answer queries, look for ground atoms.
- It is only possible to show that  $S \models \varphi$  for all *stable* models S of P:
- Show  $P \models_s \varphi$  (1st Alternative):
  - generate all stable models  $S_1, \ldots, S_n$ .
  - for each of them check whether  $S_i \models \varphi$  (if not, it is a witness for a counterexample)
- Show  $P \models_{s} \varphi$  (2nd Alternative):
  - encode  $\neg \varphi$  in a program P',
  - run smodels on  $P \cup P'$ ,
  - if there is a stable model, then,  $P \not\models_s \varphi$ (and, again, a witness for a counterexample has been found).



• two possibilities: either lokaste or Oedipus.

698

# FOL ENTAILMENT/PROOFS VS. STABLE MODELS/ASP

Consider again the italian-vs-english ontology from Slides 546 and 687:

- FOL/Tableaux: prove Spec<sub>ItalEngl</sub>  $\models \forall x : italian(x) \rightarrow lazy(x)$ (without considering any ground instance)
- ASP: show that in all stable models of italians-english.s, lazy(e) holds.
- this conclusion is weaker than the one for FOL.
  - e is a "typical" instance of an italian, but
  - recall that Stable Models reasoning is nonmonotonic more knowledge about *e* could invalidate this conclusion.

# MONOTONIC VS. NONMONOTONIC REASONING

# • FOL:

 $(\forall x : bird(x) \land \neg penguin(x) \rightarrow flies(x)) \land bird(tweety) \not\models flies(tweety)$  $(\forall x : bird(x) \land \neg penguin(x) \rightarrow flies(x)) \land bird(tweety) \not\models \neg flies(tweety)$ FOL reasoning does not entail any anything about tweety – Open World.

- $(\forall x : bird(x) \land \neg penguin(x) \rightarrow flies(x)) \land bird(tweety) \land penguin(tweety) \models \neg flies(tweety))$  $(\forall x : bird(x) \land \neg penguin(x) \rightarrow flies(x)) \land bird(tweety) \land \neg penguin(tweety) \models flies(tweety))$
- FOL is monotonic. More knowledge, more conclusions, no conclusions can/must ever be withdrawn.
- it is not possible to *conclude* things that cannot be actually proven (and that may have to be withdrawn later).

Monotonicity vs. Nonmonotonicity (cont'd) • ASP: flies(X) :- bird(X), not penguin(X). [Filename: Datalog/tweety.s] bird(tweety). Stable Model: bird(tweety) flies(tweety) bird'(tweety) flies'(tweety) flies(X) :- bird(X), not penguin(X). bird(tweety). penguin(tweety). [Filename: Datalog/tweety2.s] Stable Model: bird(tweety) penguin(tweety) bird'(tweety) penguin'(tweety)

## Monotonicity vs. Nonmonotonicity (cont'd)

This can be used to encode general concepts in nonmonotonic reasoning

- by default, if p(x) holds, and there is no further information, then usually q(x) can be assumed.
- if p(x) holds, and q(x) is consistent with the knowledge, conclude q(x):
- "Circumscription" (J. McCarthy, "Circumscription A form of non-monotonic reasoning". Artificial Intelligence 13: 27-39, April 1980).

```
flies(X) :- bird(X), not abnormal(X).
abnormal(X) :- penguin(X).
bird(tweety).
```

[Filename: Datalog/tweety-circ.s]

Stable Model: bird(tweety) flies(tweety) bird'(tweety) flies'(tweety)

- In (T, F)-notation: ({bird(tweety), flies(tweety)}, {penguin(tweety), abnormal(tweety)})
- Note that ({bird(tweety), penguin(tweety), abnormal(tweety)}, {flies(tweety) }) is also a model of P, but it is not stable.

If learning that penguin(tweety) holds, this model would become (the only) stable one.

702

# 12.2 An Application: Semantics of Referential Actions in SQL

Consider again the problem of ambiguous semantics of referential actions from Slide 236:

Country								
Name	Cod	e	Capital	Province				
Germany	Ď		Berlin	Berlin				
United States	US		Washington	Distr.Col.	C	ASCADE	Province	9
	<u> </u>					Name	Country	Capital
	S	ET	NULL	CASCADE		Berlin	D	Berlin
City				Distr.Col.	US	Washington		
Name	Count	ry	Province					
Berlin	D		В					
Washington	USA	١	Distr.Col.		DEL	ETE FROM C	ountry	
				WHERE Code='D'				

# SQL STANDARD

- The SQL Standard gives a hard-to-understand procedural specification of referential actions.
- Database Systems implemented only ON DELETE CASCADE (as optional alternative to ON DELETE NO ACTION) for a long time (late 90s)
- Nondeclarative semantics of ON DELETE SET NULL or ON UPDATE CASCADE are implemented.
   e.g. Oracle 11: the most recently defined (more exactly: activated) referential action is executed first.
- combination with transactions and PL/SQL and triggers becomes complex.

704

#### Concepts

- Intuitive concept of "Event-Condition-Action-Rules" ("ECA Rules"): ON DELETE (of referenced tuple) CASCADE (update to referencing tuple(s)
- can be read as a declarative specification how integrity is to be maintained:
   "Whenever the set U of updates includes the deletion of a referenced tuple wrt. a referential integrity constraint S.B → R.A, the cascaded update of the referencing tuple(s) must also be contained in U. (etc. for other referential actions)"
- ⇒ Set-oriented characterization of all updates that "complete" a transaction wrt. referential integrity maintenance.

... from a declarative point of view, the semantics should be easy, and it *must* be unambiguous. This lead us to playing around with Datalog (and Statelog) around Easter 1996:

#### Publications

The following papers are accessible via http://dblp.uni-trier.de/ (and partially via the DBIS Publications Web pages):

- B. Ludäscher, W. May und J. Reinert. Towards a Logical Semantics for Referential Actions in SQL. In Proc. Intl. Workshop on Foundations of Models and Languages for Data and Objects: Integrity in Databases (FMLDO'96), Dagstuhl Castle, Germany, 1996.
- B. Ludäscher, W. May und G. Lausen. Referential Actions as Logical Rules. In *Proc. ACM Symposium on Principles of Database Systems (PODS'97)*, pp. 217–224, 1997.
- B. Ludäscher und W. May. Referential Actions: From Logical Semantics to Implementation. In *Proc. Intl. Conference on Extending Database Technology (EDBT'98)*, Springer LNCS 1377, pp. 404-418, 1998.
- W. May und B. Ludäscher. Understanding the Global Semantics of Referential Actions using Logic Rules. In ACM Transactions on Database Systems (TODS), 27(4):343–397, 2002.

706

# 12.2.1 Straightforward Logical Semantics: Encoding as Rules

#### • consider only ON DELETE CASCADE/SET NULL:

% prov(Country) refs country(code) on delete cascade
<pre>del_province(PN,C,PPop,PA,PCap,PCapProv) :- del_country(N,C,Cap,CapP,A,P),</pre>
province(PN,C,PPop,PA,PCap,PCapProv).
% city(Country) refs country(code) on delete set null
upd_city(CN,C,CP,CPop,Lat,Long,El,CN,null,CP,CPop,Lat,Long,El) :-
<pre>del_country(N,C,Cap,CapP,A,P), city(CN,C,CP,CPop,Lat,Long,El).</pre>
% city(Country,Province) refs province(name,country) on delete cascade
<pre>del_city(CN,C,PN,CPop,Lat,Long,El) :- del_province(PN,C,PPop,PA,PCap,PCapProv),</pre>
<pre>city(CN,C,PN,CPop,Lat,Long,El).</pre>
<pre>inconsistent :- del_city(CN,C,CP,CPop,Lat,Long,El),</pre>
upd_city(CN,C,CP,CPop,Lat,Long,El,CN2,C2,CP2,CPop2,Lat2,Long2,E12).
country("Germany","D","Berlin","Berlin",356910,83536115).
province("Berlin","D",3472009,889,"Berlin","Berlin").
city("Berlin","D","Berlin",3472009,13,52,null).
<pre>del_country("Germany","D","Berlin","Berlin",356910,83536115).</pre>
% ?- inconsistent. [Filename: Datalog/refint.s]

cascaded updates and "inconsistent" are true.

#### Deletions: Encoding as Rules

- ON DELETE/UPDATE NO ACTION:
  - if there is a referencing tuple that is not deleted (or modified to reference another parent) in the same transaction, then the update is not allowed, e.g., in the reference Organization(City,Country,Province) → City(City,Country,Province) ON DELETE NO ACTION ON UPDATE CASCADE
    - a city where an organization has its headquarter cannot be deleted (i.e., when the city is merged with another one, the value must also be changed in the referenced organization tuple in the same transaction),
    - \* if a city where an organization has its headquarter is renamed or its province changes, then, the update is cascaded to the headquarter foreign key.
- $\Rightarrow$  all potential (cascaded) updates during the transaction must be considered.
  - ext\_ACTION ("external"): the updates issued by the user/by the program.
  - pot\_ACTION ("potential"): all updates issued by the user/by the program or resulting from these by any referential action.

708

# 12.2.2 Deletions: Encoding as Rules

- Collect all potential updates that are triggered by the external operations:
- Consider only deletions and ON DELETE CASCADE/NO ACTION.

#### Deletions: Encoding as Rules

710

# Deletions: Encoding as Rules

- Organization(City,Country,Province)  $\rightarrow$  City(City,Country,Province) ON DELETE NO ACTION ON UPDATE CASCADE
- block deletions/updates if ON UPDATE NO ACTION and child tuple remains,
- propagate blocking upwards through CASCADEs.

```
blk_del_city(Ci,Co,P,CPop,Lat,Long,El) :- pot_del_city(Ci,Co,P,CPop,Lat,Long,El),
organization(0,N,Ci,Co,P,E), rem_organization(0,N,Ci,Co,P,E).
rem_organization(0,N,Ci,Co,P,E) :- organization(0,N,Ci,Co,P,E),
not del_organization(0,N,Ci,Co,P,E). %%% del not yet defined
blk_del_province(P,C,PPop,PA,PCap,PCapProv) :- pot_del_city(CN,C,P,CPop,Lat,Long,El),
blk_del_city(CN,C,P,CPop,Lat,Long,El), pot_del_province(P,C,PPop,PA,PCap,PCapProv).
blk_del_country(N,C,Cap,CapP,A,Pop) :- pot_del_province(P,C,PPop,PA,PCap,PCapProv),
blk_del_province(P,C,PPop,PA,PCap,PCapProv), pot_del_country(N,C,Cap,CapP,A,Pop).
```

Deletions: Encoding as Rules

• (if any update is blocked do nothing).

• execute (and appropriately cascade) all external updates that are not blocked.



```
712
```

Deletions Example: delete Germany (not referenced)
country("Germany", "D", "Berlin", "Berlin", 356910, 83536115).
province("Berlin","D",3472009,889,"Berlin","Berlin").
city("Berlin","D","Berlin",3472009,13,52,null).
organization("EU","European Union","Brussels","B","Brabant","1992-02-07").
<pre>isMember("D","EU","member").</pre>
<pre>ext_del_country("Germany", "D", "Berlin", "Berlin", 356910, 83536115).</pre>
% lparse -n 0 refint1.s refint2.s refint3.s refint4.s refint-del-d.s  smodels
[Filename: Datalog/refint-del-d.s]

In this case, there is only one stable model (i.e., it coincides with the well-founded model) which is total:

```
pot_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_province("Berlin","D",3472009,889,"Berlin","Berlin")
pot_del_city("Berlin","D","Berlin",3472009,13,52,null)
pot_del_isMember("D","EU","member")
del_country("Germany","D","Berlin","Berlin",356910,83536115)
del_province("Berlin","D",3472009,889,"Berlin","Berlin")
del_city("Berlin","D","Berlin",3472009,13,52,null)
del_isMember("D","EU","member")
rem_organization("EU","European Union","Brussels","B","Brabant","1992-02-07")
```

```
Deletions Example - delete Belgium (referenced by the EU)
country("Belgium", "B", "Brussels", "Brabant", 30510, 10170241).
city("Brussels", "B", "Brabant", 951580, null, null, null).
province("Brabant", "B", 2253794, 3358, "Brussels", "Brabant").
organization("EU", "European Union", "Brussels", "B", "Brabant", "1992-02-07").
isMember("B","EU","member").
ext_del_country("Belgium", "B", "Brussels", "Brabant", 30510, 10170241).
% lparse -n 0 refint1.s refint2.s refint3.s refint4.s refint-del-b1.s| smodels
[Filename: Datalog/refint-del-b1.s]
Again, a total unique stable model = well-founded model – it contains blockings:
ext_del_country("Belgium", "B", "Brussels", "Brabant", 30510, 10170241)
pot_del_country("Belgium","B","Brussels","Brabant",30510,10170241)
pot_del_isMember("B","EU","member")
pot_del_province("Brabant", "B", 2253794, 3358, "Brussels", "Brabant")
pot_del_city("Brussels", "B", "Brabant", 951580, null, null, null)
rem_organization("EU", "European Union", "Brussels", "B", "Brabant", "1992-02-07")
                                                                                     <<<<
blk_del_city("Brussels", "B", "Brabant", 951580, null, null, null)
                                                                                     <<<<
blk_del_province("Brabant", "B", 2253794, 3358, "Brussels", "Brabant")
                                                                                     <<<<
blk_del_country("Belgium","B","Brussels","Brabant",30510,10170241)
                                                                                     <<<<
```

```
714
```

# Deletions Example - delete Belgium and the EU ext\_del\_country("Belgium", "B", "Brussels", "Brabant", 30510, 10170241). ext\_del\_organization("EU", "European Union", "Brussels", "B", "Brabant", "1992-02-07"). country("Belgium", "B", "Brussels", "Brabant", 30510, 10170241). city("Brussels", "B", "Brabant", 951580, null, null, null). province("Brabant", "B", 2253794, 3358, "Brussels", "Brabant"). organization("EU", "European Union", "Brussels", "Brabant", "1992-02-07"). isMember("B", "EU", "member"). % lparse -n 0 refint1.s refint2.s refint3.s refint4.s refint-del-b2.s| smodels [Filename: Datalog/refint-del-b2.s] Again, a total unique stable model = well-founded model: ext\_del\_country("Belgium" "B", "Brussels", "Brabant", 30510, 10170241).

```
ext_del_country("Belgium","B","Brussels","Brabant",30510,10170241)
ext_del_organization("EU","European Union","Brussels","B","Brabant","1992-02-07")
pot_del_ [...]
del_country("Belgium","B","Brussels","Brabant",30510,10170241)
del_city("Brussels","B","Brabant",951580,null,null,null)
del_province("Brabant","B",2253794,3358,"Brussels","Brabant")
del_isMember("B","EU","member")
del_organization("EU","European Union","Brussels","B","Brabant","1992-02-07")
```

# Deletions Example with NO ACTION

 Change the reference from City to Province from CASCADE to NO ACTION: (in refint1.s and refint4.s)

```
pot_del_country(N,C,Cap,CapP,A,Pop) :- ext_del_country(N,C,Cap,CapP,A,Pop),
        country(N,C,Cap,CapP,A,Pop).
% Province(Country) refs Country(code) on delete cascade
pot_del_province(P,C,PPop,PA,PCap,PCapProv) :- pot_del_country(N,C,Cap,CapP,A,Pop),
        province(P,C,PPop,PA,PCap,PCapProv).
% City(Country) refs Country(code) on delete cascade
pot_del_city(CN,C,P,CPop,Lat,Long,El) :-
    pot_del_country(N,C,Cap,CapP,A,Pop),
    city(CN,C,P,CPop,Lat,Long,El).
% City(Country, Province) refs Province(name, country) on delete no action <<<<<<
blk_del_province(P,C,PPop,PA,PCap,PCapProv) :-
  pot_del_province(P,C,PPop,PA,PCap,PCapProv),
  city(CN,C,P,CPop,Lat,Long,El), rem_city(CN,C,P,CPop,Lat,Long,El).
rem_city(CN,C,P,CPop,Lat,Long,El) :- city(CN,C,P,CPop,Lat,Long,El),
                               not del_city(CN,C,P,CPop,Lat,Long,El).
[Filename: Datalog/refint1b.s]
```

```
716
```

Deletions: Encoding as Rules
<pre>del_country(N,C,Cap,CapP,A,Pop) :- ext_del_country(N,C,Cap,CapP,A,Pop),</pre>
<pre>country(N,C,Cap,CapP,A,Pop), not blk_del_country(N,C,Cap,CapP,A,Pop).</pre>
<pre>del_province(P,C,PPop,PA,PCap,PCapProv) :- pot_del_country(N,C,Cap,CapP,A,Pop),</pre>
<pre>del_country(N,C,Cap,CapP,A,Pop), province(P,C,PPop,PA,PCap,PCapProv).</pre>
<pre>del_city(CN,C,P,CPop,Lat,Long,El) :- pot_del_country(N,C,Cap,CapP,A,Pop),</pre>
<pre>del_country(N,C,Cap,CapP,A,Pop), city(CN,C,P,CPop,Lat,Long,El).</pre>
<pre>del_isMember(C,O,T) :- pot_del_country(N,C,Cap,CapP,A,P),</pre>
<pre>del_country(N,C,Cap,CapP,A,P), isMember(C,O,T).</pre>
<pre>del_organization(0,N,Ci,Co,P,E) :- ext_del_organization(0,N,Ci,Co,P,E),</pre>
<pre>organization(0,N,Ci,Co,P,E), not blk_del_organization(0,N,Ci,Co,P,E).</pre>
<pre>del_isMember(C,0,T) :- del_organization(0,N,Ci,Co,P,E), isMember(C,0,T).</pre>
[Filename: Datalog/refint4b.s]

 using pot\_del is sometimes redundant, and only required for providing smodels with a domain predicate.

#### **Deletions: The Resulting Models**

% lparse -n 0 refint1b.s refint2.s refint3.s refint4b.s refint-del-d.s| smodels

• 3 stable models, 2 of them total, one partial ( = well-founded model).

```
ext_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_province("Berlin","D",3472009,889,"Berlin","Berlin")
pot_del_city("Berlin","D","Berlin",3472009,13,52,null)
pot_del_province("Berlin","D",3472009,889,"Berlin","Berlin")
del_country("Germany","D","Berlin","Berlin",356910,83536115)
del_city("Berlin","D","Berlin",3472009,13,52,null)
ext_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_[...]
blk_del_country("Germany","D","Berlin","Berlin",356910,83536115)
blk_del_province("Berlin","D",3472009,889,"Berlin","Berlin","Berlin")
rem_city("Berlin","D","Berlin",3472009,13,52,null)
```

• in the partial stable model (=the well-founded model), all pot\_del are true, all blk\_del and all del are undefined.

#### Deletions: Interpreting the Resulting Models

3 stable models:

- 1. the well-founded, partial one: all pot\_del are true, the blk\_del and del are undefined.
  - two total stable ones:
    - 2. all pot\_del are true, the del are true, and the blk\_del are false (since the reason for blk\_del\_prov(...) is deleted).
    - 3. all pot\_del are true, the blk\_del are true, and the del are false (since the deletion of the province is not allowed due to blocking).

#### Application-Specific Priorities (cf. EDBT paper)

- the "intended" one is the stable model (2) that gives priority to deletions against blockings.
- including game-theoretic interpretation:
  - "del\_p( $\bar{x}$ )" is won if deletion is possible
  - counter-moves: "how?" and "what about this referencing tuple  $q(\bar{y})$ ",
  - justification "cascaded from "del\_r( $\bar{z}$ )" and "claim: del\_q( $\bar{y}$ )",
  - infinite games (cycling around via deleted blocking tuples) are won for the deleter.
  - the well-founded model can be used  $\rightarrow$  polynomial.
#### Deletions: Encoding as Rules

- Sets of (external) delete requests are monotonic: Let *del*(*D*) denote the actual set of (cascaded) deletions on the database.
- $del(D_1 \cup D_2) = del(D_1) \cup del(D_2)$ ,
- If D<sub>1</sub> and D<sub>2</sub> are admissible (wrt. NO ACTION), then D<sub>1</sub> ∪ D<sub>2</sub> is admissible (but not vice versa, recall the "Belgium" example – D<sub>1</sub> ∪ D<sub>2</sub> is admissible even though D<sub>1</sub> alone is not admissible).
- with CASCADE and SET NULL, conflicts can arise;
- with DELETE+CASCADE/SET NULL and UPDATE even more conflicts can arise.

720

## 12.2.3 Updates: Encoding as Rules

- with updates, the modeling is more involved:
  - updates can create new foreign keys,
  - usually such updates are cascaded from the appropriate parent,
  - but overlapping FKs induce interferences with other K/FKs.
  - Consider S.X $\rightarrow$ R.Y ON UPDATE NO ACTION and r(a), r(b) and s(a). A transaction that modifies  $r(a) \rightarrow r(c)$  and  $s(a) \rightarrow s(b)$  is admissible.
  - $\Rightarrow$  not only tuple-based, but key-foreign-key-based.
  - different cascaded modifications can be applied at the same time to a tuple.
- Recall:
  - SET NULL cannot create new foreign keys null values cannot violate any SQL integrity constraint (except NOT NULL).
  - actual conflicts in a "diamond" from a single update can only result from CASCADE and SET NULL since then *different* changes are applied.
- the approach must cover the general "worst case", not only "intuitive" cases.

#### Updates: Encoding as Rules

- for every key and foreign key: pot\_chg\_R\_K\_1...K\_k(X\_1,...,X\_n,Y\_1,...,Y\_k) and chg\_R\_K\_1...K\_k(X\_1,...,X\_n,Y\_1,...,Y\_k) (change f/key attributes K\_1,...,K\_k of tuple R(X\_1,...,X\_n) to Y\_1,...,Y\_k).
- propagation for every FK/K reference (ON UPDATE CASCADE): pot\_prp\_R<sub>P</sub>\_R<sub>C</sub>\_F<sub>1</sub>...F<sub>k</sub>(X<sub>1</sub>,...,X<sub>n</sub>,Y<sub>1</sub>,...,Y<sub>k</sub>) (propagate update from R<sub>P</sub>'s keys to FK F<sub>1</sub>,...,F<sub>k</sub> of tuple R<sub>C</sub>(X<sub>1</sub>,...,X<sub>n</sub>)).
- user updates: projection to the keys and foreign keys (K<sub>1</sub>, ..., K<sub>k</sub>) of R: pot\_prp\_ext\_R\_K<sub>1</sub>...K<sub>k</sub>(X<sub>1</sub>,...X<sub>n</sub>,Y<sub>i1</sub>,...,Y<sub>ik</sub>) :ext\_mod\_R(X<sub>1</sub>,...,X<sub>n</sub>,Y<sub>1</sub>,...,Y<sub>n</sub>), (Y<sub>i1</sub>,...,Y<sub>ik</sub>) != (X<sub>i1</sub>,...,X<sub>ik</sub>).
- collect propagated changes to keys/foreign keys (overlappings!): Simplified pattern: Consider two "incoming" propagations from R<sub>P1</sub>.A → R<sub>C</sub>.K<sub>1</sub> and R<sub>P1</sub>.B → R<sub>C</sub>.K<sub>2</sub> concern the key (K<sub>1</sub>, K<sub>2</sub>) of R<sub>C</sub>: pot\_chg\_R<sub>C</sub>\_K<sub>1</sub>,K<sub>2</sub>(X<sub>1</sub>,...,X<sub>n</sub>,Y<sub>1</sub>,Y<sub>2</sub>) :pot\_prp\_R<sub>P1</sub>\_R<sub>C</sub>\_K<sub>1</sub>(X<sub>1</sub>,...,X<sub>n</sub>,Y<sub>1</sub>), pot\_prp\_R<sub>P2</sub>\_R<sub>C</sub>\_K<sub>2</sub>(X<sub>1</sub>,...,X<sub>n</sub>,Y<sub>2</sub>), (Y<sub>1</sub>,Y<sub>2</sub>) != (X<sub>K1</sub>,X<sub>K2</sub>). (for the fully general rule see (CH<sub>1</sub>) in the TODS paper)

### Updates: Encoding as Rules

Changes of Primary Keys  $R_P(K_1, ..., K_k)$  are then handled according to the referential actions of their "child tuples"

•  $R_C(F_1, \ldots, F_k)$  REFERENCES  $R_P(K_1, \ldots, K_k)$  ON UPDATE NO ACTION: blk chg  $R_P$   $K_1, ..., K_k(X_1, ..., X_n, Y_1, ..., Y_k)$ :pot chg  $R_P$   $K_1,\ldots,K_k(X_1,\ldots,X_n,Y_1,\ldots,Y_k)$ , rem refd  $R_P$   $R_C$   $F_1\ldots,F_k(Y_1,\ldots,Y_k)$ . •  $R_C(F_1, \ldots, F_k)$  REFERENCES  $R_P(K_1, \ldots, K_k)$  ON UPDATE CASCADE: pot\_prp\_ $R_P_R_C_F_1...F_k(Z_1,...,Z_n,Y_1,...,Y_k)$ :pot chg  $R_P$   $K_1, ..., K_k(X_1, ..., X_n, Y_1, ..., Y_k)$ ,  $\pi[\mathsf{F}_1,\ldots,\mathsf{F}_k](\mathsf{Z}_1,\ldots,\mathsf{Z}_n) = \pi[\mathsf{K}_1,\ldots,\mathsf{K}_k](\mathsf{X}_1,\ldots,\mathsf{X}_n).$ and (block propagation if change of child is blocked) blk prp  $R_P R_C F_1...F_k(X_1,...,X_n,Y_1,...,Y_k)$ :pot\_prp\_R<sub>P</sub>\_R<sub>C</sub>\_F<sub>1</sub>...F<sub>k</sub>(X<sub>1</sub>,...,X<sub>n</sub>,Y<sub>1</sub>,...,Y<sub>k</sub>), blk chg  $\mathsf{R}_C$   $\mathsf{F}_1,\ldots,\mathsf{K}_k(\mathsf{X}_1,\ldots,\mathsf{X}_n,\mathsf{Y}_1,\ldots,\mathsf{Y}_k)$ and (block parent change if propagation to some child is blocked) blk chg  $R_P$   $K_1, ..., K_k(X_1, ..., X_n, Y_1, ..., Y_k)$ , pot chg  $R_P$   $K_1, ..., K_k(X_1, ..., X_n, Y_1, ..., Y_k)$ , blk\_prp\_R<sub>P</sub>\_R<sub>C</sub>\_F<sub>1</sub>...F<sub>k</sub>(Z<sub>1</sub>,...,Z<sub>n</sub>,Y<sub>1</sub>,...,Y<sub>k</sub>),  $\pi[\mathsf{F}_1,\ldots,\mathsf{F}_k](\mathsf{Z}_1,\ldots,\mathsf{Z}_n) = \pi[\mathsf{K}_1,\ldots,\mathsf{K}_k](\mathsf{X}_1,\ldots,\mathsf{X}_n).$ 

Updates: Encoding as Rules

Further rules (see TODS paper):

- when a key value remains referenced (child not deleted, not "modified away"),
- when a key value gets newly referenced (insert, "modified towards"),
- when a key value gets newly referencable (insert, "modified towards"),
- when a change of a FK is blocked because the new reference does not exist,
- when 2 changes on a tuple are inconsistent (del/upd, different values (null vs. a value),
- rules that derive the modifications to be finally executed.

#### Results

- if an update set is admissible, the well-founded model is sufficient: giving priorities to modifications/propagations/changes vs. blockings yields a total stable model and the updates to be executed
- if an update set is not admissible, the stable models indicate what portions of the initial set are admissible, and where the problems are located (mutually excluding updates, missing cascades, unresolved tuples).

724

# 12.3 The Limits

Recall:

 ASP is based on atoms (i.e., disjunction is not dealt with as formula, but resolved into atoms)

(note: in tableaux, it is also broken down to atoms)

- ASP algorithms are internally based on ground atoms and grounding of all rules (with all ground terms=constants of the Herbrand universe).
  (note: tableaux can keep variables and use them on-demand)
- ASP is closely related with Model Checking: generate models by a strategy.

#### Expressiveness

- Datalog<sup>¬</sup> rules, for ASP extended with:
  - disjunction/choice rules: generate suitable different models (cf. tableau: branches)
  - negation: via denials. Discard models.
    (cf. tableau: close branch)

#### The Limits

The Limits of ASP:

- no existential quantification, object invention:
  - every person has a father who is again a person. (tableau: via skolemization and strategical application of the  $\delta$ -rule)
  - ASP/grounding: would require infinitely many ground instances.
    (tableau: strategic application of the γ-Rule + Blocking strategies (e.g., DL/OWL reasoning))
- *P* ⊨ *a* means only that *a* holds in the minimal/stratified/well-founded model.
  Statements about ¬*a* only by default negation/closed world.
  (On the other hand: LP provides a CWA reasoning formalism in contrast to tableaux)

#### Comparison: Tableaux

- Tableaux can be tailored to any logic: FOL, DL/OWL, ...
  - Open-World: monotonic
  - complex expansion & blocking strategies + heuristics.
- ASP is worst-case exponential, but with a polynomial basis: the AFP computation for the well-founded model.

726

# 12.4 DB vs. KB: Closed World vs. Open World

Consider the following formula *F*:

 $F \equiv person("John", 35) \land person("Alice", 10) \land person("Bob", 8) \land \land person("Carol", 12) \land person("Jack", 65) \land \land child("John", "Alice") \land child("John", "Bob") \land$ 

 $\forall X, Y : (\exists Z : (child(Z, X) \land child(Z, Y) \land X \neq Y) \rightarrow sibling(X, Y))$ 

- Does *child*("John", "Bob") hold? obviously yes.
- Does G:=sibling("Alice", "Bob") hold?
  - (Relational) Database: *sibling* is a view. The answer is "yes".
  - FOL KB: for all models  $\mathcal{M}$  of F, G holds. Thus,  $F \models sibling(\text{"Alice"}, \text{"Bob"})$ .
- What about G:=sibling("Alice", "Carol")?
  - (Relational) Database: no. For the database state  $\mathbf{D}, \mathbf{D} \not\models sibling$ ("Alice", "Carol").
  - FOL KB: there is a model  $\mathcal{M}_1$  of F, where  $\mathcal{M}_1 \not\models G$ , but there is also a model  $\mathcal{M}_2$  of F, where  $\mathcal{M}_2 \models G$  (e.g., add the tuple ("John", "Carol") to the interpretation of *child*).

For the Web, *child*("John", "Carol") can e.g. be contributed by another Web Source.

## **DB** vs. KB: Closed World vs. Open World

- What about G := child("John", "Jack")?
  - (Relational) Database: no. For the database state  $\mathbf{D}, \mathbf{D} \not\models child$ ("John", "Jack").
  - FOL KB: there is a model  $\mathcal{M}_1$  of F, where  $\mathcal{M}_1 \not\models G$ , but there is also a model  $\mathcal{M}_2$  of F, where  $\mathcal{M}_2 \models G$ .
- Obviously, the KB does not know that a child cannot be older than its parents.
  Add a constraint to *F*, obtaining *F*':

 $F' :\equiv F \land \forall P, C, A_1, A_2, : (person(P, A_1) \land person(C, A_2) \land child(P, C)) \rightarrow A_1 > A_2$ 

- database: this assertion would prevent to add *child*("John", "Jack") to the database.

- for the KB,  $F' \models \neg child$ ("John", "Jack") allows to *infer* that Jack is not the child of John. Such information can be given with the *ontology* of a domain.

728

## DB vs. KB: CLOSED WORLD vs. OPEN WORLD

- the Database Model Theory is called "*Closed World*": things that are not known to hold are *assumed* not to hold.
- the FOL semantics is called *"Open World"*: things that are not known to be true or false are considered to be *possible*.

### **CONSEQUENCES ON NEGATION**

- in databases there is no explicit negation. It is not necessary to specify that Jack is *not* a child of John.
- in a KB, it would be necessary to state ... ∧ ¬*child*("John", X) for all persons who are known not to be children of John.
  Additional constraints: extend the ontology, e.g., by stating that a person has exactly two parents then all others cannot be parents works only for persons whose parents are known. Similarly for the "age" constraint from the previous slide.
- note that the semantics of universal quantification (∀) is also effected: ∀X : φ is equivalent to ¬∃X : ¬φ.

## **REASONING IN PRESENCE OF NEGATION**

Obtaining new information (e.g., by finding another Web Source) has different effects on Open vs. Closed world:

 Closed world: conclusions drawn before – "Carol is not a child of John", or "John has exactly two children" from less information can become invalid.

This kind of reasoning is *nonmonotonic* 

• Open world: everything which is not known explicitly is taken into account to be possible (by considering all possible models).

This kind of reasoning is *monotonic*:

 $\mathsf{Knowledge}_1 \subseteq \mathsf{Knowledge}_2 \Rightarrow \mathsf{Conclusions}_1 \subseteq \mathsf{Conclusions}_2$ 

 Open World can be combined with other forms of nonmonotonic reasoning, e.g., Defaults: "usually, birds can fly". Knowing that Tweety is a bird allows to conclude that it flies.
 Obtaining the information that Tweety is a penguin (which can usually not fly) leads to invalidation of this conclusion.

The current Semantic Web research mainstream prefers Open World without default reasoning.

730

COMPARISON, MOTIVATION ETC.						
Database vs. FOL						
Relational Databases	relational schema		tuples	SQL queries		closed world
FOL	signature (predicates +functions)		facts (atoms)	$\mathcal{S} \models \phi$ ? (yes/no or answer $\psi \models \phi$ ? variable bindings)		mostly: open world sometimes closed world
Situations and tasks						
Given		what to do		how?		
facts/database		does $p(\ldots)$ hold in the DB? SQL query		by combining data		
facts+constraints (SQL assertions or FOL formulas)		additionally: test if constraints satisfied		equivalent to first situation (query for violating tuples)		
facts (DB) rules (KB)		does $p(\ldots)$ hold in DB+rules?		DB+views application of rules		
facts (DB) knowledge base KB as FOL formulas		is a formula $\phi$ entailed by DB+KB?			reasoning, entailment, KB $\models \phi$ ?	