

# Relational Query Languages

Universidad de Concepción, 2014

(Slides adapted from Loreto Bravo, who adapted from Werner Nutt who adapted them  
from Thomas Eiter and Leonid Libkin)

Bases de Datos II

1

## Databases

A database is

- a collection of structured data
- along with a set of access and control mechanisms

We deal with them every day:

- back end of Web sites
- telephone billing
- bank account information
- e-commerce
- airline reservation systems, store inventories, library catalogs, . . .

## Data Models: Ingredients

- Formalisms to represent information (schemas and their instances), e.g.,
  - relations containing tuples of values
  - trees with labeled nodes, where leaves contain values
  - collections of triples (subject, predicate, object)
- Languages to query represented information, e.g.,
  - relational algebra, first-order logic, Datalog, Datalog<sup>⌈</sup>
  - tree patterns
  - graph pattern expressions
  - SQL, XPath, SPARQL

Bases de Datos II

3

- Languages to describe changes of data (updates)

Relational Query Languages

## Questions About Data Models and Queries

Given a schema  $\mathcal{S}$  (of a fixed data model)

- is a given structure (FOL interpretation, tree, triple collection)  
an instance of the schema  $\mathcal{S}$ ?
- does  $\mathcal{S}$  have an instance at all?

Given queries  $Q, Q'$  (over the same schema)

- what are the answers of  $Q$  over a fixed instance  $I$ ?
- given a potential answer  $a$ , is  $a$  an answer to  $Q$  over  $I$ ?
- is there an instance  $I$  where  $Q$  has an answer?
- do  $Q$  and  $Q'$  return the same answers over all instances?

Relational Query Languages

**Questions About Query Languages**

Given query languages  $\mathcal{L}$ ,  $\mathcal{L}'$

- how difficult is it for queries in  $\mathcal{L}$ 
  - to evaluate such queries?
  - to check satisfiability?
  - to check equivalence?
- for every query  $Q$  in  $\mathcal{L}$ , is there a query  $Q'$  in  $\mathcal{L}'$  that is equivalent to  $Q$ ?

**Research Questions About Databases**

Relational Query Languages

- Incompleteness, uncertainty
  - How can we represent incomplete and uncertain information?
  - How can we query it? . . . and what should be the meaning of an answer?
- Information integration
  - How can we query many independent databases simultaneously?
  - How do we represent their contents? . . . and the relationships between them?
- Data streams
  - What is a good language for querying rapidly changing data?
- Concurrency control
  - How should we coordinate access to data?

## Formalization of Relational Databases

1. Relational Data Model
  - (a) Named Perspective
  - (b) Unnamed Perspective
2. Logic Programming Approach
3. First-Order Logic Approach

### 1.(a) The Relational Data Model: Named Perspective

Relational Query Languages

- Data is organized in relations (“tables”)
- A relational database **schema** consists of
  - a set of **relation names**
  - a list of **attributes** for each relation
- Notation: `<relation name>: <list of attributes>`
- Examples:
  - `Account: number, branch, customerId`
  - `Movie: title, director, actor`
  - `Schedule: theater, title`
- Relations have different names
- Attributes *within a relation* have different names

Relational Query Languages

Example: Relational Database

Movie	title	director	actor
	Shining	Kubrick	Nicholson
	Player	Altman	Robbins
	Chinatown	Polanski	Nicholson
	Chinatown	Polanski	Polanski
	Repulsion	Polanski	Deneuve

Schedule	theater	title
	Le Champo	Shining
	Le Champo	Chinatown
	Le Champo	Player
Bases de Datos II	Odéon	Chinatown
	Odéon	Repulsion

Formal Definitions

We assume three disjoint countably infinite sets of *symbols*:

- **att**, the possible *attributes*  
...we assume there is a total ordering “ $\leq_{\mathbf{att}}$ ” on **att**
- **dom**, the possible *constants*  
**dom** is called the *domain*
- **relname**, the possible *relation names*

Relations have a *sort* and an *arity*, formalized as follows:

- For every relation name  $R$  there is a finite set of attributes  $sort(R)$ .  
That is,  $sort$  is a function

$$sort: \mathbf{relname} \rightarrow \mathcal{P}^{fin}(\mathbf{att})$$

We assume as well:  $sort^{-1}(U)$  is infinite, for each  $U \in \mathcal{P}^{fin}(\mathbf{att})$

- The *arity* of a relation is the number of attributes:  $arity(R) = |sort(R)|$
- Notation: Often  $R[U]$  where  $U = sort(R)$ , or  
 $R: A_1, \dots, A_n$  if  $sort(R) = \{A_1, \dots, A_n\}$  and  $A_1 \leq_{att} \dots \leq_{att} A_n$ .

Example:  $sort(Account) = \{number, branch, customerId\}$   
is denoted  $Account: number, branch, customerId$

Relations and databases have schemas:

- A *relation schema* is a relation name
- A *database schema* **R** is a nonempty finite set of relation schemas

Example: Database schema **C** = { Account, Movie, Schedule }

Account: number, branch, customerId

Movie: title, director, actor

Schedule: theater, title

Tuples

- A *tuple* is a function

$$t: U \rightarrow \mathbf{dom}$$

mapping a finite set  $U \subseteq \mathbf{att}$  (a sort) to constants.

Example: Tuple  $t$  on  $sort(Movie)$  such that

$$\begin{aligned} t(\mathbf{title}) &= \mathbf{Shining} \\ t(\mathbf{director}) &= \mathbf{Kubrick} \\ t(\mathbf{actor}) &= \mathbf{Nicholson} \end{aligned}$$

- For  $U = \emptyset$ , there is only one tuple: the empty tuple, denoted  $\langle \rangle$
- If  $U \subseteq V$ , then  $t[V]$  is the restriction of  $t$  to  $V$

Example:

$$\langle \mathbf{title} : \mathbf{Shining}, \mathbf{director} : \mathbf{Kubrick}, \mathbf{actor} : \mathbf{Nicholson} \rangle$$

## Instances of Relations and Databases

- A *relation* or *relation instance* of a relation schema  $R[U]$  is a finite set of tuples on  $U$
- A *database instance* of database schema  $\mathbf{R}$  is a mapping  $\mathbf{I}$  that assigns to each  $R \in \mathbf{R}$  a relation instance

## 1.(b) The Relational Model: Unnamed Perspective

Alternative view: We ignore names of attributes, relations have only arities

- Tuples are elements of a Cartesian product of **dom**
- A tuple  $t$  of arity  $n \geq 0$  is an element of  $\mathbf{dom}^n$ , for example

$$t = \langle \text{Shining}, \text{Kubrick}, \text{Nicholson} \rangle$$

- We access components of tuples via their position  $i \in \{1, \dots, n\}$ :

$$t(2) = \text{Kubrick}$$

- Note: Because of “ $\leq_{\text{att}}$ ”, unnamed and named perspective naturally correspond

## 2. Logic Programming Perspective

- A *fact* over relation  $R$  with arity  $n$  is an expression  $R(a_1, \dots, a_n)$ , where  $a_1, \dots, a_n \in \mathbf{dom}$ .
- A relation (instance) is a finite set of facts over  $R$
- A database instance  $\mathbf{I}$  of  $\mathbf{R}$  is the union of relation instances for each  $R \in \mathbf{R}$

Example:

$\mathbf{I} = \{ \text{Movie}(\text{Shining}, \text{Kubrick}, \text{Nicholson}), \text{Movie}(\text{Player}, \text{Altman}, \text{Robbins}),$   
 $\text{Movie}(\text{Chinatown}, \text{Polanski}, \text{Nicholson}),$   
 $\text{Movie}(\text{Chinatown}, \text{Polanski}, \text{Polanski}),$   
 $\text{Movie}(\text{Repulsion}, \text{Polanski}, \text{Deneuve}), \text{Schedule}(\text{Le Champo}, \text{Shining}),$   
 $\text{Schedule}(\text{Le Champo}, \text{Chinatown}), \text{Schedule}(\text{Le Champo}, \text{Player}),$   
 $\text{Schedule}(\text{Odeon}, \text{Chinatown}), \text{Schedule}(\text{Odeon}, \text{Repulsion}) \}$

Bases de Datos II

15

## 3. First-Order Logic: Database Instances as Theories

Relational Query Languages

- For a database instance  $\mathbf{I}$ , construct an *extended relational theory*  $\Sigma_{\mathbf{I}}$  consisting of:
  - *Atoms*  $R_i(\vec{a})$  for each  $\vec{a} \in \mathbf{I}(R_i)$ ;
  - *Extension Axioms*  $\forall \vec{x} (R_i(\vec{x}) \leftrightarrow \vec{x} = \vec{a}_1 \vee \dots \vee \vec{x} = \vec{a}_m)$ , where  $\vec{a}_1, \dots, \vec{a}_m$ , are all elements of  $R_i$  in  $\mathbf{I}$ , and “=” ranges over tuples of the same arity;
  - *Unique Name Axioms*:  $\neg(c_i = c_j)$  for each pair  $c_i, c_j$  of distinct constants occurring in  $\mathbf{I}$ ;
  - *Domain Closure Axiom*:  $\forall x (x = c_1 \vee \dots \vee x = c_n)$ , where  $c_1, \dots, c_n$  is a listing of all constants occurring in  $\mathbf{I}$ .
- If the “=” are not available, the intended meaning can be emulated with equality axioms.
- **Theorem:** The interpretations of  $\mathbf{dom}$  and  $\mathbf{R}$  that satisfy  $\Sigma_{\mathbf{I}}$  are isomorphic to  $\mathbf{I}$
- **Corollary:** A set of sentences  $\Gamma$  is satisfied by  $\mathbf{I}$  iff  $\Sigma_{\mathbf{I}} \cup \Gamma$  is satisfiable.

Other view: database instance  $\mathbf{I}$  as *finite relational structure*

(finite universe of discourse; considered later)

Relational Query Languages



Database Queries: Examples

- “What are the titles of current movies?”

answer	title
	Shining
	Player
	Chinatown
	Repulsion

- “Which theaters are showing movies directed by Polanski?”

answer	theater
	Le Champo
	Odéon

- “Which theaters are showing movies featuring Nicholson?”

answer	theater
	Le Champo
	Odéon

- “Which directors acted themselves?”

answer	director
	Polanski

- “Who are the directors whose movies are playing in all theaters?”

answer	director
	Polanski

- “Which theaters show only movies featuring Nicholson?”

answer	theater

... but if Le Champo stops showing 'Player', the answer contains 'Le Champo'.

### How Ask a Query over a Relational Database?

- Query languages
  - Commercial: SQL
  - Theoretical: Relational Algebra, Relational Calculus, datalog etc.
- Query results: Relations constructed from relations in the database

### Declarative vs Procedural

- In our queries, we ask **what** we want to see in the output . . .
- . . . but we do not say **how** we want to get this output.
- Thus, query languages are **declarative**: they specify what is needed in the output, but do not say how to get it.
- A query engine figures out **how** to get the result,  
and gives it to the user.
- A query engine operates internally with an **algebra** that takes into account how data is stored.
- Finally, queries in that algebra are translated into a **procedural** language.

Declarative vs Procedural: Example

Declarative:

$\{ \text{title} \mid (\text{title}, \text{director}, \text{actor}) \in \text{Movie} \}$

Procedural:

```
for each tuple T=(t,d,a) in relation Movie do

    output t

end
```

Query Languages

Relational Query Languages

1. SQL

```
SELECT Schedule.Theater
FROM   Schedule, Movie
WHERE  Movie.Title = Schedule.Title AND Movie.Director = 'Polanski'
```

2. Relational calculus

$\{x \mid \exists y z (\text{Movie}(y,\text{'Polanski'},z) \wedge \text{Schedule}(x,y))\}$

3. Rule-based queries

$\text{answer}(x) \text{ :- Movie}(y,\text{'Polanski'},z), \text{Schedule}(x,y)$

4. Relational Algebra

$\pi_{\text{Schedule.Theater}}(\sigma_{\text{Movie.Title=Schedule.Title} \wedge \text{Movie.Director='Polanski'}}(\text{Movie} \times \text{Schedule}))$

*Are these query languages equivalent?*

- To queries  $q_1$  and  $q_2$  over a schema  $\mathbf{R}$  are *equivalent* ( $q_1 \equiv q_2$ ) if  $q_1(\mathbf{I}) = q_2(\mathbf{I})$  for each instance  $\mathbf{I}$  over  $\mathbf{R}$ .
- Let  $\mathcal{Q}_1$  and  $\mathcal{Q}_2$  be two query languages:
  - $\mathcal{Q}_1$  is *dominated* by  $\mathcal{Q}_2$  ( $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ ) if for each query  $q_1 \in \mathcal{Q}_1$  there exists a  $q_2 \in \mathcal{Q}_2$  such that  $q_1 \equiv q_2$ .
  - $\mathcal{Q}_1$  and  $\mathcal{Q}_2$  are *equivalent* ( $\mathcal{Q}_1 \equiv \mathcal{Q}_2$ ) if  $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$  and  $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ .

In what follows we will introduce several query languages and show which ones are equivalent

### Conjunctive Queries

- Conjunctive queries are a simple form of declarative, *rule-based queries*
- A *rule* says *when* certain elements belong to the answer.
- **Example:** “What are the titles of current movies?”

As a conjunctive query:

$\text{answer}(\text{tl}) \text{ :- Movie}(\text{tl}, \text{dir}, \text{act})$

That is, while  $(\text{tl}, \text{dir}, \text{act})$  ranges over relation *Movies*, output *tl* (the title attribute)

**Conjunctive Queries: One More Example**

“Which theaters are showing movies directed by Polanski?”

As a conjunctive query:

$\text{answer}(\text{th}) \text{ :- Movie}(\text{tl}, \text{'Polanski'}, \text{act}), \text{Schedule}(\text{th}, \text{tl})$

While (tl, dir, act) range over tuples in Movie

if dir is 'Polanski'

look at all tuples (th, tl) in Schedule

corresponding to the title tl of the tuple in the relation Movie

and output th.

**Conjunctive Queries: Another Example**

“Which theaters are showing movies featuring Nicholson?”

Very similar to the previous example:

$\text{answer}(\text{th}) \text{ :- Movie}(\text{tl}, \text{dir}, \text{'Nicholson'}), \text{Schedule}(\text{th}, \text{tl})$

Conjunctive queries are probably the most **common** type of queries  
and are **building blocks** for all other queries over relational databases.

Conjunctive Queries: Still One More ...

“Which directors acted in one of their own movies?”:

answer(dir) :- Movie(tl, dir, act), dir=act

While (tl, dir, act) ranges over tuples in movie,  
check if dir is the same as act,  
and output it if that is the case.

Alternative formulation:

answer(dir) :- Movie(tl, dir, dir)

Conjunctive Queries: Definition

A rule-based conjunctive query with (in)equalities is an expression of form

answer( $\vec{x}$ ) :-  $R_1(\vec{x}_1), \dots, R_n(\vec{x}_n)$ , (1)

where  $n \geq 0$  and

- “answer” is a relation name not in  $\mathbf{R} \cup \{ =, \neq \}$
- $R_1, \dots, R_n$  are relation names from  $\mathbf{R} \cup \{ =, \neq \}$
- $\vec{x}$  is a tuple of distinct variables with length = arity(answer)
- $\vec{x}_1, \dots, \vec{x}_n$  are tuples of variables and constants of suitable (?) length
- each variable occurring somewhere in the query **must** also occur in some atom  $R_i(\vec{x}_i)$  where  $R_i \in \mathbf{R}$

Note: Equality “=” can be eliminated if we change the definition slightly

How?

## Conjunctive Queries: Semantics

Let  $q$  be a conjunctive query of the form (1) and let  $\mathbf{I}$  be a database instance.

- A *valuation*  $\nu$  over  $var(q)$  is a mapping

$$\nu: var(q) \cup \mathbf{dom} \rightarrow \mathbf{dom}$$

that is the identity on  $\mathbf{dom}$ .

- The *result* (aka image) of  $q$  on  $\mathbf{I}$  is

$$q(\mathbf{I}) = \{ \nu(\vec{x}) \mid \nu \text{ is a valuation over } var(q), \text{ and } \nu(\vec{x}_i) \in \mathbf{I}(R_i), \text{ for all } 1 \leq i \leq n \}$$

**Example:**  $q: \text{answer}(\text{dir}) :- \text{Movie}(\text{tl}, \text{dir}, \text{act}), \text{dir}=\text{act}$

For  $\mathbf{I}$  from above, we obtain

$$q(\mathbf{I}) = \{ \langle \text{Polanski} \rangle \}$$

## Elementary Properties of Conjunctive Queries

**Proposition.** Let  $q$  be a conjunctive query of form (1). Then:

- the result  $q(\mathbf{I})$  is *finite*, for any database instance  $\mathbf{I}$ ;
- $q$  is *monotonic*,  
i.e.,  $\mathbf{I} \subseteq \mathbf{J}$  implies  $q(\mathbf{I}) \subseteq q(\mathbf{J})$ , for all database instances  $\mathbf{I}$  and  $\mathbf{J}$ ;
- if  $q$  contains neither “=” nor “ $\neq$ ”, then  $q$  is *satisfiable*,  
i.e., there exists some  $\mathbf{I}$  such that  $q(\mathbf{I}) \neq \emptyset$

Beyond Conjunctive Queries?

“Who are the directors whose movies (not necessarily all of them) are playing in all theaters?”

- Recall the notation from mathematical logic:  
 $\forall$  means ‘for all’,  $\exists$  means ‘exists’, “ $\wedge$ ” is conjunction (logical ‘and’)
- We write the query above as

$$\{ \text{dir} \mid \forall \text{th} ( \exists \text{tl}' (\text{Schedule}(\text{th}, \text{tl}') \rightarrow \exists \text{tl}, \text{act} (\text{Movie}(\text{tl}, \text{dir}, \text{act}) \wedge \text{Schedule}(\text{th}, \text{tl}))) ) \}$$

- That is, to see if director *dir* is in the answer, for each theater name *th*, check that there exists a tuple (*tl*, *dir*, *act*) in *Movie*, and a tuple (*th*, *tl*) in *Schedule*

Bases de Datos II

31

*Can we formulate this as a conjunctive query?*

Structured Query Language: SQL

Relational Query Languages

- De-facto standard for all relational RDBMs
- Latest versions: SQL:1999 (also called SQL3), SQL:2003 (supports XML), SQL:2006 (more XML support), SQL:2008  
Each standard covers well over 1,000 pages

”The nice thing about standards is that you have so many to choose from.”  
– Andrew S. Tanenbaum.

- Query structure:

SELECT     $R_{i_1}.A_{j_1}, \dots, R_{i_k}.A_{j_k}$  (attribute list)  
FROM        $R_1, \dots, R_n$   
WHERE       $C$  (condition)

In the simplest case,  $C$  is a conjunction of equalities/inequalities

Relational Query Languages



**SQL Examples**

- “Which theaters are showing movies directed by Polanski?”:

```
SELECT Schedule.Theater
FROM   Schedule, Movie
WHERE  Movie.Title = Schedule.Title AND
       Movie.Director = 'Polanski'
```

- “Which theaters are playing the movies of which directors?”

```
SELECT Movie.Director, Schedule.Theater
FROM   Movie, Schedule
WHERE  Movie.Title = Schedule.Title
```

Bases de Datos II

33

**Relational Algebra (Named Perspective)**

Relational Query Languages

- We start with a subset of relational algebra that suffices to capture queries defined by

simple rules,

SQL SELECT-FROM-WHERE statements

- The subset has three operations:

*Projection*  $\pi$

*Selection*  $\sigma$

*Cartesian Product*  $\times$

- This fragment of Relational Algebra is called *SPC Algebra*
- Sometimes we also use *renaming* of attributes, denoted as  $\rho$

Relational Query Languages

Projection

- Restricts tuples of a relation  $R$  to a subset of  $sort(R)$
- $\pi_{A_1,\dots,A_n}(R)$  returns a new relation with sort  $\{ A_1, \dots, A_n \}$
- Example:

$\pi_{title,director}$ 

title	director	actor
Shining	Kubrick	Nicholson
Player	Altman	Robbins
Chinatown	Polanski	Nicholson
Chinatown	Polanski	Polanski
Repulsion	Polanski	Deneuve

=

title	director
Shining	Kubrick
Player	Altman
Chinatown	Polanski
Repulsion	Polanski

- Creates a *view* of the original data ~~that hides some attributes~~

Selection

- Chooses tuples of  $R$  that satisfy some condition  $C$
- $\sigma_C(R)$  returns a new relation with the same sort as  $R$ ,  
and with the tuples  $t$  of  $R$  for which  $C(t)$  is true
- Conditions are conjunctions of *elementary conditions* of the form
$$R.A = R.A' \text{ (equality between attributes)}$$
$$R.A = constant \text{ (equality between an attribute and a constant)}$$
same as above but with  $\neq$  instead of  $=$
- Examples:
$$Movie.Actor = Movie.Director$$
$$Movie.Actor = Movie.Director \wedge Movie.Actor \neq 'Nicholson'$$
- Creates a *view* of data by hiding tuples that do not satisfy the condition

Selection: Example

$\sigma_{actor=director \wedge director='Polanski'}$

title	director	actor
Shining	Kubrick	Nicholson
Player	Altman	Robbins
Chinatown	Polanski	Nicholson
Chinatown	Polanski	Polanski
Repulsion	Polanski	Deneuve

=

title	director	actor
Chinatown	Polanski	Polanski

Cartesian Product

Relational Query Languages

- $R_1 \times R_2$  is a relation with  $sort(R_1 \times R_2) = sort(R_1) \cup sort(R_2)$  and the tuples are all possible combinations  $(t_1, t_2)$  of  $t_1$  in  $R_1$  and  $t_2$  in  $R_2$

- Example:

$R_1$	$A$	$B$	$R_2$	$A$	$C$	$R_1.A$	$R_1.B$	$R_2.A$	$R_2.C$
	$a_1$	$b_1$		$a_1$	$c_1$	$a_1$	$b_1$	$a_1$	$c_1$
	$a_2$	$b_2$		$a_2$	$c_2$	$a_1$	$b_1$	$a_2$	$c_2$
				$a_3$	$c_3$	$a_1$	$b_1$	$a_3$	$c_3$
						$a_2$	$b_2$	$a_1$	$c_1$
						$a_2$	$b_2$	$a_2$	$c_2$
						$a_2$	$b_2$	$a_3$	$c_3$

- We assume that the cartesian product operator automatically renames attributes so as to include the name of the relation: in the resulting table, all attributes must have different names.

Relational Query Languages

Cartesian Product: Example

“Which theaters are playing movies directed by Polanski?”

$\text{answer}(\text{th}) \text{ :- Movie}(\text{tl}, \text{dir}, \text{act}), \text{Schedule}(\text{th}, \text{tl}), \text{dir} = \text{'Polanski'}$

- Step 1: Let  $R_1 = \text{Movie} \times \text{Schedule}$

We don’t need all tuples, only those in which titles are the same, so:

- Step 2: Let  $R_2 = \sigma_C(R_1)$  where  $C$  is “Movie.title = Schedule.title”

We are only interested in movies directed by Polanski, so

- Step 3:  $R_3 = \sigma_{\text{director} = \text{'Polanski'}}(R_2)$

In the output, we only want theaters, so finally

- Step 4: Answer =  $\pi_{\text{theater}}(R_3)$

- Summing up, the answer is

$$\pi_{\text{theater}}(\sigma_{\text{director} = \text{'Polanski'}}(\sigma_{\text{Movie.title} = \text{Schedule.title}}(\text{Movie} \times \text{Schedule})))$$

- Merging selections, this is equivalent to

$$\pi_{\text{theater}}(\sigma_{\text{director} = \text{'Polanski'} \wedge \text{Movie.title} = \text{Schedule.title}}(\text{Movie} \times \text{Schedule})))$$

Renaming

- Let  $R$  be a relation that has attribute  $A$  but does *not* have attribute  $B$ .
- $\rho_{B \leftarrow A}(R)$  is the “same” relation as  $R$  except that  $A$  is renamed to be  $B$ .

Example:

$\rho_{\text{parent} \leftarrow \text{father}}$

father	child
George	Elizabeth
Philip	Charles
Charles	William

=

parent	child
George	Elizabeth
Philip	Charles
Charles	William

- Simultaneous renaming  $\rho_{A_1, \dots, A_m \leftarrow B_1, \dots, B_m}$ , for distinct  $A_1, \dots, A_m$  resp.  $B_1, \dots, B_m$  can be defined from it.

- Prefixing the relation name to rename attributes is convenient (used in practice)
- Not all problems are solved by this (e.g., Cartesian Product  $R \times R$ )

Relational Algebra in the Unnamed Perspective

- The same as before, except for Renaming, which becomes immaterial      *Why?*
- Example (again): “Which theaters are playing movies directed by Polanski?”

Recall

Movie: title, director, actor

Schedule: theater, title

$\pi_4(\sigma_{2='Polanski' \wedge 1=5}(\text{Movie} \times \text{Schedule}))$

- SPC Algebra is often assumed to be based in the unnamed setting
- Other operations of Relational Algebra can only be defined for named perspective (e.g., natural join, to be seen later)

SQL and Relational Algebra

For execution, declarative queries are translated into algebra expressions

- Idea:       SELECT is projection  $\pi$   
              FROM    is Cartesian product  $\times$   
              WHERE   is selection  $\sigma$

- A simple case (only one relation in FROM):

```
SELECT  A,B,...
FROM    R
WHERE   C
```

is translated into        $\pi_{A,B,...}(\sigma_C(R))$

Translating Declarative Queries into Relational Algebra

We use rules as intermediate format

**Example:** “Which are the titles of movies?”

- SELECT Title  
  FROM   Movie
- answer(tl) :- Movie(tl,dir,act)
- $\pi_{\text{title}}(\text{Movie})$

... this was simply projection

A More Elaborate Translation Example

“Which theaters are showing movies directed by Polanski?”

- `SELECT Schedule.Theater`  
`FROM Schedule, Movie`  
`WHERE Movie.Title = Schedule.Title AND`  
`Movie.Director='Polanski'`
- First, translate into a rule:  
`answer(th) :- Schedule(th,tl), Movie(tl,'Polanski',act)`
- Second, change the rule such that:  
constants appear only in conditions  
no variable occurs twice

- This gives us:  
`answer(th) :- Schedule(th,tl), Movie(tl,dir,act), dir = 'Polanski', tl=tl`

A More Elaborate Translation Example (cntd)

`answer(th) :- Schedule(th,tl), Movie(tl',dir,act), dir = 'Polanski', tl=tl'`

Two relations  $\implies$  Cartesian product

Conditions  $\implies$  selection

Subset of attributes in the answer  $\implies$  projection

- Step 1:  $R_1 = \text{Schedule} \times \text{Movie}$
- Step 2: Make sure we talk about the same movie:

$$R_2 = \sigma_{\text{Schedule.title}=\text{Movie.title}}(R_1)$$

- Step 3: We are only interested in Polanski’s movies:

$$R_3 = \sigma_{\text{Movie.director}=\text{Polanski}}(R_2)$$

- Step 4: We need only theaters in the output

$$\text{answer} = \pi_{\text{Schedule.theater}}(R_3)$$

A More Elaborate Translation Example (cntd)

Summing up, the answer is:

$$\pi_{\text{Schedule.theater}}\Big(\sigma_{\text{Movie.director}=\text{Polanski}}\Big(\sigma_{\text{Schedule.title}=\text{Movie.title}}\Big(\text{Schedule}\times\text{Movie}\Big)\Big)\Big)$$

or, using the rule  $\sigma_{C_1}(\sigma_{C_2}(R)) = \sigma_{C_1\wedge C_2}(R)$ :

$$\pi_{\text{Schedule.theater}}\Big(\sigma_{\text{Movie.director}=\text{Polanski} \wedge \text{Schedule.title}=\text{Movie.title}}\Big(\text{Schedule}\times\text{Movie}\Big)\Big)$$

Formal Translation: SQL to Rules

SELECT    attribute list  $\langle R_i.A_j \rangle$   
FROM        $R_1, \dots, R_n$   
WHERE      condition  $C$

is translated into:

$$\begin{aligned} \text{answer}(\langle R_i.A_j \rangle) \quad &:- \quad R_1(\text{<attributes>}), \\ &\dots, \\ &R_n(\text{<attributes>}), \\ &C \end{aligned}$$

Note: Attributes become variables of rules



**Rules to Relational Algebra**

- Consider the rule

$$\text{answer}(\vec{x}) :- R_1(\vec{x}_1), \dots, R_n(\vec{x}_n) \quad (2)$$

where, wlog (= “without loss of generality”),

$$R_1, \dots, R_k \in \mathbf{R}, k \leq n,$$

$$R_{k+1}, \dots, R_n \in \{=, \neq\}.$$

Let  $\text{conditions} := R_{k+1}(\vec{x}_{k+1}), \dots, R_n(\vec{x}_n)$

- First transformation:* Ensure that each variable occurs at most once in  $R_1(\vec{x}_1), \dots, R_k(\vec{x}_k)$ :

If there are  $R_i(\dots, x, \dots)$  and  $R_j(\dots, x, \dots)$ ,

rewrite them as  $R_i(\dots, x', \dots)$  and  $R_j(\dots, x'', \dots)$ , and

add  $x' = x''$  to the conditions and, if  $x$  occurs elsewhere, also  $x = x'$

**Example:**

Relational Query Languages

$\text{answer}(\text{th}, \text{dir}) :- \text{movie}(\text{tl}, \text{dir}, \text{act}), \text{schedule}(\text{th}, \text{tl})$

is rewritten to

$\text{answer}(\text{th}, \text{dir}) :- \text{movie}(\text{tl}', \text{dir}, \text{act}), \text{schedule}(\text{th}, \text{tl}'), \text{tl}' = \text{tl}$

- Next step:* each occurrence of a constant  $a$  in a relational atom  $R_i(\dots, a, \dots)$ ,  $R_i \in \mathbf{R}$ , is replaced by some variable  $x$  and add  $x = a$  to the conditions
- Finally:* after the rule (2) is rewritten, it is translated into

$$\pi_{\hat{x}}(\sigma_{\widehat{\text{conditions}}}(R_1 \times \dots \times R_k))$$

where  $\hat{\cdot}$  maps

- a variable  $x$  occurring in some  $R_i(\dots, x, \dots)$ ,  $R_i \in \mathbf{R}$ ,  
to the corresponding attribute  $\hat{x}$  in  $\text{sort}(R_i)$ ;
- an expression  $\alpha$  to the expression  $\hat{\alpha}$  where every  $x$  is replaced by  $\hat{x}$

Relational Query Languages

Putting it Together: SQL to Relational Algebra

Combine the two translation steps:

SQL  $\mapsto$  rule-based queries  
rule-based queries  $\mapsto$  relational algebra.

This yields the following translation from SQL to relational algebra:

```
SELECT  attribute list  $\langle R_i.A_j \rangle$ 
FROM     $R_1, \dots, R_n$ 
WHERE   condition  $C$ 
```

becomes

Bases de Datos II

$$\pi_{\langle R_i.A_j \rangle}(\sigma_C(R_1 \times \dots \times R_n))$$

51

Relational Query Languages

Another Example

“Which theaters show movies featuring Nicholson?”

```
SELECT Schedule.Theater
FROM   Schedule, Movie
WHERE  Movie.Title = Schedule.Title
      AND Movie.Actor='Nicholson'
```

- Translate into a rule:

answer(th) :- movie(tl, dir, 'Nicholson'), schedule(th, tl)

- Modify the rule:

answer(th) :- movie(tl, dir, act), schedule(th, tl'), tl=tl', act='Nicholson'

Relational Query Languages

answer(th) :- movie(tl, dir, act), schedule(th, tl'), tl=tl', act='Nicholson'

- Step 1:  $R_1 = \text{Schedule} \times \text{Movie}$
- Step 2: Make sure we talk about the same movie:

$$R_2 = \sigma_{\text{Schedule.title}=\text{Movie.title}}(R_1)$$

- Step 3: We are only interested in movies with Nicholson:

$$R_3 = \sigma_{\text{Movie.actor}=\text{Nicholson}}(R_2)$$

- Step 4: we need only theaters in the output

$$\text{answer} = \pi_{\text{schedule.theater}}(R_3)$$

Summing up:

$$\pi_{\text{schedule.theater}}(\sigma_{\text{Movie.actor}=\text{Nicholson} \wedge \text{Schedule.title}=\text{Movie.title}}(\text{Schedule} \times \text{Movie}))$$

SPC Algebra into SQL

Should be easy, but is it?

*Where's the difficulty?*

- Direct proof in two steps:

Show that for SPC queries there are normal forms

$$\pi_{A_1,\dots,A_n}(\sigma_c(R_1 \times \dots \times R_m)),$$

called “simple SPC queries”      *(proof idea?)*

Then map normal forms to SQL

- Indirect proof:

SPC is equivalent to conjunctive queries

Conjunctive queries are equivalent to single block SQL queries

Extension: Natural Join

- Combine all pairs of tuples  $t_1$  and  $t_2$  in relations  $R_1$  resp.  $R_2$  that agree on common attributes
- The resulting relation  $R = R_1 \bowtie R_2$  is the **natural join** of  $R$  and  $S$ , defined on the *set* of attributes in  $R_1$  and  $R_2$ .

Example: Schedule  $\bowtie$  Movie

title	director	actor		theater	title		title	director	actor	theater
Shining	Kubrick	Nicholson		Le Champo	Shining		Shining	Kubrick	Nicholson	Le Champo
Player	Altman	Robbins		Le Champo	Chinatown		Player	Altman	Robbins	Le Champo
Chinatown	Polanski	Nicholson	$\bowtie$	Le Champo	Player	$=$	Chinatown	Polanski	Nicholson	Le Champo
Chinatown	Polanski	Polanski		Odéon	Chinatown		Chinatown	Polanski	Nicholson	Odéon
Repulsion	Polanski	Deneuve		Odéon	Repulsion		Chinatown	Polanski	Polanski	Le Champo
							Chinatown	Polanski	Polanski	Odéon
							Repulsion	Polanski	Deneuve	Odéon

Bases de Datos II55

Natural Join cont'd

Natural join is not a new operation of relational algebra

- It is **definable** with  $\pi, \sigma, \times$  (*and renaming!?*)
- Suppose
  - $R$  is a relation with attributes  $A_1, \dots, A_n, B_1, \dots, B_k$
  - $S$  is a relation with attributes  $A_1, \dots, A_n, C_1, \dots, C_m$ $\implies R \bowtie S$  has attributes  $A_1, \dots, A_n, B_1, \dots, B_k, C_1, \dots, C_m$
- Then

$R \bowtie S =$

$\pi_{A_1, \dots, A_n, B_1, \dots, B_k, C_1, \dots, C_m}(\sigma_{R.A_1=S.A_1 \wedge \dots \wedge R.A_n=S.A_n}(R \times S))$

Could a natural join be defined in the unnamed perspective?

Select Project Join Queries (SPJ Queries)

Queries of the form

$$\pi_{A_1,\dots,A_n}(\sigma_c(R_1 \bowtie \dots \bowtie R_m))$$

are called Select-project-join queries.

- These are probably the most common queries (over databases with foreign keys).

**Example:** “Which theaters show movies directed by Polanski?”

- `answer(th) :- schedule(th,tl), movie(tl,'Polanski',act)`
- As SPJ query:

$$\pi_{\text{theater}}(\sigma_{\text{director}=\text{'Polanski'}}(\text{Movie} \bowtie \text{Schedule}))$$

- Why has the query become simpler compared to the earlier version

Bases de Datos II

57

$$\pi_{\text{schedule.theater}}(\sigma_{\text{Movie.director}=\text{'Polanski'} \wedge \text{Schedule.title}=\text{Movie.title}}(\text{Schedule} \times \text{Movie}))?$$

Relational Query Languages

SPJ Queries cont'd

“Which theaters show movies featuring Nicholson?”

- As rule-based conjunctive query

$$\text{answer(th)} \text{ :- movie(tl, dir, 'Nicholson'), schedule(th, tl)}$$

- As SPJ query:

$$\pi_{\text{theater}}(\sigma_{\text{actor}=\text{'Nicholson'}}(\text{Movie} \bowtie \text{Schedule}))$$

Relational Query Languages

Translating SPJ Queries to Rules and Single Block SQL

- SPJ Query

$$Q = \pi_{A_1, \dots, A_n}(\sigma_C(R \bowtie S))$$

- Equivalent SQL statement ( $B_1, \dots, B_m$  = common attributes in  $R$  and  $S$ ):

```
SELECT  A1, ..., An
FROM    R, S
WHERE   C AND R.B1 = S.B1 AND ... AND R.Bm = S.Bm
```

- Equivalent rule query ( $R$  resp.  $S$  has attributes:  $C_1, \dots, C_k$  resp.  $D_1, \dots, D_l$ )

Bases de Datos II

$$\text{answer}(A_1, \dots, A_n) \text{ :- } R(C_1, \dots, C_k), S(D_1, \dots, D_l), \\ R.B_1 = S.B_1, \dots, R.B_m = S.B_m, C$$

59

SPJ to SQL: Example

“Who are the directors of currently playing movies that feature Ford?”

- In SPJ:

$$\pi_{\text{director}}(\sigma_{\text{actor} = \text{'Ford'}}(\text{Movie} \bowtie \text{Schedule}))$$

- In SQL:

```
SELECT  Movie.director
FROM    Movie, Schedule
WHERE   Movie.title = Schedule.title AND
        Movie.actor = 'Ford'
```

### What We've Seen So Far

- Queries defined by SQL `SELECT-FROM-WHERE` statements (single block queries)
- These are the same as the queries definable by rules
- They are also the same as the queries definable by  $\pi, \sigma, \times$  (and renaming) in relational algebra, i.e., the same as SPC queries
- Question: What about SPJ?

SPJ queries are *not* a normal form for the  $\sigma, \pi, \times$ -fragment

$\leadsto$  To prevent unwanted joins, we need renaming

- **SPJR Algebra** =  $\sigma, \pi, \bowtie, \rho$  — fragment of Relational Algebra

61

Relational Query Languages

### Equivalence of SPC and SPJR Algebras

**Proposition.** The SPC Algebra and the SPJR Algebra are equivalent.

Note:

- Cartesian Product can be easily emulated using renaming
- BTW, also SQL provides a renaming construct

New attribute names can be introduced in `SELECT` using keyword `AS`.

```
SELECT Father AS Parent, Child
FROM R
```

Relational Query Languages

**Nested SQL Queries: Simple Example**

- So far in the `WHERE` clause we used comparisons between attributes
- In general, a `WHERE` clause can contain *another query*, and test some relationship between an attribute or a constant and the result of that query
- We call such queries with subqueries *nested* queries

**Example:** “Which theaters are showing Polanski’s movies?”

```
SELECT Schedule.theater
FROM   Schedule
WHERE  Schedule.title IN
      (SELECT Movie.title
       FROM Movie
       WHERE Movie.director = 'Polanski')
```

**Nested vs Unnested Queries**

<pre>SELECT S.theater FROM   Schedule S WHERE  S.title IN       (SELECT M.title        FROM Movie M        WHERE M.director = 'Polanski')</pre>	<pre>SELECT S.theater FROM   Schedule S, Movie M WHERE  S.title = M.title AND       M.director = 'Polanski'</pre>
---	---

- Both queries capture the same question ...
- ... and return the same results over all instances (... *or do they?*)
- Queries nested with `IN` can be flattened ...
- ... but others can't (*which?*)



Equivalence Theorem

**Theorem.** The following languages define the same (?) sets of queries:

- SPJR Queries
- SPC Queries
- simple SPC queries
- (rule-based) conjunctive queries
- SQL SELECT-FROM-WHERE
- SQL SELECT-FROM-WHERE with IN-nesting

Disjunction in Queries

“Which actors played in movies directed by Kubrick *OR* Polanski”

- ```
SELECT Actor
FROM   Movie
WHERE  director = 'Kubrick' OR director = 'Polanski'
```
- Can this be defined by a *single* rule?
- How do you prove your answer?  
*(Hint: What can you say about the constants in the query and in the database?)*

Union in SQL

- The way out: Disjunction can be represented using more than one rule

answer(act) :- movie(tl,dir,act), dir='Kubrick'  
answer(act) :- movie(tl,dir,act), dir='Polanski'

- Semantics: compute answers to each of the rules, and then take their *union* (*union of conjunctive queries*)
- SQL has its own syntax (distinguishing between UNION and UNION ALL):

```
SELECT Actor
FROM   Movie
WHERE  director = 'Kubrick'
UNION
SELECT Actor
FROM   Movie
WHERE  director = 'Polanski'
```

Disjunction in Relational Algebra

How can we translate a query with disjunction into relational algebra?

- answer(act) :- movie(tl,dir,act), dir='Kubrick'
- is translated into

$$Q_1 = \pi_{actor}(\sigma_{director=Kubrick}(Movie))$$

- answer(act) :- movie(tl,dir,act), dir='Polanski'
- is translated into

$$Q_2 = \pi_{actor}(\sigma_{director=Polanski}(Movie))$$

- The whole query is translated into  $Q_1 \cup Q_2$ , i.e.,

$$\pi_{actor}(\sigma_{director=Kubrick}(Movie)) \cup \pi_{actor}(\sigma_{director=Polanski}(Movie))$$

## Union in Relational Algebra

- Union is another operation of relational algebra

$R \cup S$  is the union of relations  $R$  and  $S$

$R$  and  $S$  must have the same set of attributes (be “union-compatible”).

- We now have four relational algebra operations:

$$\pi, \sigma, \times, \cup$$

(and of course  $\bowtie$ , which is definable from  $\pi, \sigma, \times$ )

- This fragment is called the SPCU-Algebra, or *positive relational algebra*.

*Would an intersection operator add something new?*

Bases de Datos II

*And what about set difference?*

Relational Query Languages

## Identities Among Relational Algebra Operators

- $\pi_{A_1, \dots, A_n}(R \cup S) = \pi_{A_1, \dots, A_n}(R) \cup \pi_{A_1, \dots, A_n}(S)$
- $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
- $(R \cup S) \times T = R \times T \cup S \times T$
- $T \times (R \cup S) = T \times R \cup T \times S$

Relational Query Languages

### Normal Form of SPCU Queries

**Theorem.** *Every SPCU query is equivalent to a union of SPC queries*

Proof: propagate the union operation.

Example:

$$\begin{aligned}
 & \pi_A(\sigma_c((R \times (S \cup T)) \cup W)) \\
 &= \pi_A(\sigma_c((R \times S) \cup (R \times T) \cup W)) \\
 &= \pi_A(\sigma_c(R \times S) \cup \sigma_c(R \times T) \cup \sigma_c(W)) \\
 &= \pi_A(\sigma_c(R \times S)) \cup \pi_A(\sigma_c(R \times T)) \cup \pi_A(\sigma_c(W))
 \end{aligned}$$

Bases de Datos II

71

### Another Equivalence Theorem

Relational Query Languages

**Theorem.** The following languages define the same sets of queries

- Positive relational algebra (SPCU queries)
- unions of SPC queries
- queries defined by multiple rules
- unions of conjunctive queries
- SQL SELECT-FROM-WHERE-UNION
- SQL SELECT-FROM-WHERE-UNION with IN-nesting
- SPJRU queries ( $\sigma, \pi, \bowtie, \rho, \cup$ )

*Would intersection add anything new?*

Relational Query Languages