Foundations of Databases

Relational Query Languages

Free University of Bozen-Bolzano, 2009

Werner Nutt

(Slides adapted from Thomas Eiter and Leonid Libkin)

Databases

A database is

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

We deal with them every day:

- bank account information,
- airline reservation systems,
- store inventories,
- library catalogs,
- telephone billing etc.

Goals of a Database Management System

- Provide users with a meaningful **view** of data:

  Hide from them irrelevant detail $\longrightarrow$ provide an abstract view of data

- Support various **operations** on data

  queries: getting answers from databases

  updates: changing information in databases

- **Coordinate** access to data

  concurrency control

The Relational Data Model: Named Perspective

- 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

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
	Odéon	Chinatown
	Odéon	Repulsion

Formal Definitions

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

- $\mathbf{att}$, the possible *attributes*

  ...we assume there is a total ordering "$\leq_{\mathbf{att}}$" on $\mathbf{att}$

- $\mathbf{dom}$, the possible *constants*

  $\mathbf{dom}$ is called the *domain*

- $\mathbf{relname}$, 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

$$sort\colon \mathbf{relname} \to \mathcal{P}^{fin}(\mathbf{att})$$

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

What does this mean?

- The *arity* of a relation is the number of attributes: $arity(R) = |sort(R)|$
- Notation: Often $R[U]$ where $U = sort(R)$, or

  $R\colon A_1, \ldots, A_n$ if $sort(R) = \{A_1, \ldots, A_n\}$ and $A_1 \leq_{\mathbf{att}} \cdots \leq_{\mathbf{att}} A_n$.

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

  is denoted `Account: number, branch, customerId`

Relations and databases have schemas:

- A *relation schema* is a relation name
- A *database schema* $\mathbf{R}$ 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

$$t\colon U \to \mathbf{dom}$$

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

  Example: Tuple t on $sort(\texttt{Movie})$ such that

$$\begin{array}{rcl} t(\texttt{title}) & = & \texttt{Shining} \\ t(\texttt{director}) & = & \texttt{Kubrick} \\ t(\texttt{actor}) & = & \texttt{Nicholson} \end{array}$$

- 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:

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

The Relational Model: Unnamed Perspective

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

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

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

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

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

- Note: Because of "$\leq_{\mathbf{att}}$", unnamed and named perspective naturally correspond

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.

$\rightsquigarrow$ Other perspectives:

Logic programming p.

First-order logic p.

Logic Programming Perspective

- A *fact* over relation R with arity n is an expression $R(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n \in \mathbf{dom}$.
- 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:

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

First-Order Logic: Database Instances as Theories

- 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* $\forall \vec{x}(R_i(\vec{x}) \leftrightarrow \vec{x} = \vec{a}_1 \vee \cdots \vee \vec{x} = \vec{a}_m)$, where $\vec{a}_1, \ldots \vec{a}_m$, 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:* $\forall x(x = c_1 \vee \cdots \vee x = c_n)$, where $c_1, \ldots, c_n$ 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 Γ 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)

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 To Ask a Query?

- 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 database management system figures out **how** to get the result, and gives it to the user.
- A database management system 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:

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

Procedural:

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


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:

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

  That is, while (tl, dir, 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:

answer(th) :– Movie(tl, 'Polanski', act), Schedule(th, tl)

While (tl, dir, act) range over tuples in Movie, check if dir is 'Polanski';
if not, go to the next tuple,
if yes, look at all tuples (th, tl) in Schedule
corresponding to the title tl in relation Movie,
and output th.

Conjunctive Queries: Another Example

"Which theaters are showing movies featuring Nicholson?"

Very similar to the previous example:

answer(th) :– Movie(tl, dir, 'Nicholson'), Schedule(th, tl)

While (tl, dir, act) range over tuples in Movie, check if act is 'Nicholson';
if not, go to the next tuple,
if yes, look at all tuples (th, tl) in Schedule
corresponding to the title tl in relation Movie,
and output th.

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

$$\text{answer}(\vec{x}) \text{ :– } R_1(\vec{x_1}), \ldots, R_n(\vec{x_n}), \tag{1}$$

where $n \geq 0$ and

- “answer” is a relation name not in $\mathbf{R} \cup \{=, \neq\}$
- $R_1, \ldots, R_n$ are relation names from $\mathbf{R} \cup \{=, \neq\}$
- $\vec{x}$ is a tuple of distinct variables with length $=$ arity$($answer$)$
- $\vec{x_1}, \ldots, \vec{x_n}$ 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* ν over $var(q)$ is a mapping

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

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

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

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

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 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

  $\{$ dir $|$ $\forall$ th ($\exists$ tl' (Schedule(th,tl') $\longrightarrow$

  $\exists$ tl, act (Movie(tl,dir,act) $\wedge$ Schedule(th, 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

Is there something missing?

Can we formulate this as a conjunctive query?

Structured Query Language: SQL

- 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` attribute list $\langle R_i.A_j \rangle$

  `FROM` $R_1, \ldots, R_n$

  `WHERE` condition C

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

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
```


Relational Algebra

- 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 π

  Selection σ

  Cartesian Product $\times$

- This fragment of Relational Algebra is called *SPC Algebra*

- Sometimes we also use *renaming* of attributes, denoted as ρ

Projection

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

$\pi_{\text{title,director}}$

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 hat 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

  $R.A = R.A'$ (equality between attributes)

  $R.A =$ *constant* (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

$$\sigma_{\text{actor}=\text{director} \wedge \text{director}='\text{Polanski}'} \left(\begin{array}{ccc} \text{title} & \text{director} & \text{actor} \\ \hline \text{Shining} & \text{Kubrick} & \text{Nicholson} \\ \text{Player} & \text{Altman} & \text{Robbins} \\ \text{Chinatown} & \text{Polanski} & \text{Nicholson} \\ \text{Chinatown} & \text{Polanski} & \text{Polanski} \\ \text{Repulsion} & \text{Polanski} & \text{Deneuve} \end{array} \right)$$

$$= \begin{array}{ccc} \text{title} & \text{director} & \text{actor} \\ \hline \text{Chinatown} & \text{Polanski} & \text{Polanski} \end{array}$$

Cartesian Product

- $R_1 \times R_2$ 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:

$$\begin{array}{c|cc} R_1 & A & B \\ \hline & a_1 & b_1 \\ & a_2 & b_2 \end{array} \times \begin{array}{c|cc} R_2 & A & C \\ \hline & a_1 & c_1 \\ & a_2 & c_2 \\ & a_3 & c_3 \end{array} = \begin{array}{cccc} R_1.A & R_1.B & R_2.A & R_2.C \\ \hline a_1 & b_1 & a_1 & c_1 \\ a_1 & b_1 & a_2 & c_2 \\ a_1 & b_1 & a_3 & c_3 \\ a_2 & b_2 & a_1 & c_1 \\ a_2 & b_2 & a_2 & c_2 \\ a_2 & b_2 & a_3 & c_3 \end{array}$$

- 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.

Cartesian Product: Example

"Which theaters are playing movies directed by Polanski?"

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

- Step 1: Let $R_1 =$ Movie $\times$ 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:

$$\rho_{\textsf{parent}\leftarrow\textsf{father}} \left(\begin{array}{cc} \text{father} & \text{child} \\ \hline \text{George} & \text{Elizabeth} \\ \text{Philip} & \text{Charles} \\ \text{Charles} & \text{William} \end{array} \right) = \left(\begin{array}{cc} \text{parent} & \text{child} \\ \hline \text{George} & \text{Elizabeth} \\ \text{Philip} & \text{Charles} \\ \text{Charles} & \text{William} \end{array} \right)$$

- Simultaneous renaming $\rho_{A_1,\ldots,A_m \leftarrow B_1,\ldots,B_m}$, for distinct $A_1, \ldots, A_m$ resp. $B_1, \ldots, B_m$ 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 $R \times R$)

Unnamed Perspective

- Renamings are for SPC immaterial, if we adopt the unnamed perspective

  Why?

- Example (again): "Which theaters are playing movies directed by Polanski?"

  Recall
```
Movie: title, director, actor
Schedule: theater, title
```

$$\pi_1(\sigma_{2='\textsf{Polanski}' \wedge 1=5}(\textsf{Movie} \times \textsf{Schedule})))$$

- 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 π
  `FROM` is Cartesian product $\times$
  `WHERE` is selection σ

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

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

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

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)

- $\pi_{\text{title}}\big(\text{Movie}\big)$

. . . 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 $\Longrightarrow$ Cartesian product

Conditions $\Longrightarrow$ selection

Subset of attributes in the answer $\Longrightarrow$ projection

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

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

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

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

- Step 4: We need only theaters in the output

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

A More Elaborate Translation Example (cntd)

Summing up, the answer is:

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

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

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

Formal Translation: SQL to Rules

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

is translated into:

answer($\langle R_i.A_j \rangle$) :- R_1(`<attributes>`),
$\ldots$,
R_n(`<attributes>`),
C

Note: Attributes become variables of rules

Rules to Relational Algebra

- Consider the rule

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

where, wlog (= "without loss of generality"),

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

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

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

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

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

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

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

Example:

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

is rewritten to

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

- *Next step:* each occurrence of a constant a in a relational atom $R_i(..., a, ...)$, $R_i \in \mathbf{R}$, is replaced by some variable x and add $x = a$ to the conditions

- *Finally:* after the rule (2) is rewritten, it is translated into

$$\pi_{\widehat{\vec{x}}}(\sigma_{\widehat{conditions}}(R_1 \times \cdots \times R_n))$$

where $\widehat{\cdot}$ maps

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

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`	$R_1, \ldots, R_n$
`WHERE`	condition C

becomes

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

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

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

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

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

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

- Step 4: we need only theaters in the output

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

Summing up:

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

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

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

  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 $\boxed{R = R_1 \bowtie R_2}$ 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
Shining	Kubrick	Nicholson
Player	Altman	Robbins
Chinatown	Polanski	Nicholson
Chinatown	Polanski	Polanski
Repulsion	Polanski	Deneuve

$\bowtie$

theater	title
Le Champo	Shining
Le Champo	Chinatown
Le Champo	Player
Odéon	Chinatown
Odéon	Repulsion

$=$

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

Natural Join cont'd

Natural join is not a new operation of relational algebra

- It is **definable** with π, σ, $\times$ *(and renaming!?)*
- Suppose
  - R is a relation with attributes $A_1, \ldots, A_n,\ B_1, \ldots, B_k$
  - S is a relation with attributes $A_1, \ldots, A_n,\ C_1, \ldots, C_m$

  $\implies$ $R \bowtie S$ has attributes $A_1, \ldots, A_n,\ B_1, \ldots, B_k, C_1, \ldots, C_m$
- Then

$$R \bowtie S =$$

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

Could a natural join be defined in the unnamed perspective?

Select Project Join Queries (SPJ Queries)

Queries of the form

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

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:

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

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

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

SPJ Queries cont’d

“Which theaters show movies featuring Nicholson?”

- As rule-based conjunctive query

answer(th) :– movie(tl, dir, ’Nicholson’), schedule(th, tl)

- As SPJ query:

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

Translating SPJ Queries to Rules and Single Block SQL

- SPJ Query

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

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

  SELECT $A_1, \ldots, A_n$

  FROM R, S

  WHERE C AND $R.B_1 = S.B_1$ AND ... AND $R.B_m = S.B_m$

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

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

SPJ to SQL: Example

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

- In SPJ:

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

- 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 π, σ, $\times$ (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 σ, π, $\times$-fragment

  $\leadsto$ To prevent unwanted joins, we need renaming
- SPJR Algebra = σ, π, $\bowtie$, ρ — fragment of Relational Algebra

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
```

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

```
SELECT S.theater                    SELECT S.theater
FROM   Schedule S                   FROM   Schedule S, Movie M
WHERE  S.title IN                   WHERE  S.title = M.title AND
   (SELECT M.title                         M.director = ’Polanski’
    FROM   Movie M
    WHERE  M.director = ’Polanski’)
```

- 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 `UINON` 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

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

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

  is translated into

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

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

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

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?

And what about set difference?

Identities Among Relational Algebra Operators

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

Normal Form of SPCU Queries

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

Proof: propagate the union operation.

Example:

$$\pi_A(\sigma_c((R \times (S \cup T)) \cup W))$$

$$= \pi_A(\sigma_c((R \times S) \cup (R \times T) \cup W))$$

$$= \pi_A(\sigma_c(R \times S) \cup \sigma_c(R \times T) \cup \sigma_c(W))$$

$$= \pi_A(\sigma_c(R \times S)) \cup \pi_A(\sigma_c(R \times T)) \cup \pi_A(\sigma_c(W)$$

Another Equivalence Theorem

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?