Foundations of Databases

Relational Query Languages

Free University of Bozen – Bolzano, 2005

Thomas Eiter

Institut für Informationssysteme

Arbeitsbereich Wissensbasierte Systeme (184/3)

Technische Universität Wien

`http://www.kr.tuwien.ac.at/staff/eiter`

Databases

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

We deal with them every day:

- airline reservation systems,
- personnel directories,
- store inventories,
- bank account information, etc etc

Goals of a Database System

- Provide users with a meaningful view of data:

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

- Support various operations on data

  queries: getting answers from databases

  updates: changing information in databases

- Control data

  concurrency control

The Relational Data Model

- Data is organized in relations (tables)
- Relational database schema:

    set of table names

    list of attributes for each table

- Tables are specified as: `<table name>:<list of attributes>`
- Examples:

```
Account: number, branch, customerId
Movie: title, director, actor
Schedule: theater, title
```

- Attributes within a table have different names
- Tables 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

- $\mathbf{att}$ ... countably infinite set of *attributes*

  Assumption: total ordering $\leq_{\mathbf{att}}$ on $\mathbf{att}$

- $\mathbf{dom}$ ... countably infinite set called *domain*

  Elements of $\mathbf{dom}$ are called *constants*

- $\mathbf{relname}$ ... countably infinite set of *relation names*

$\mathbf{dom}$, $\mathbf{att}$ and $\mathbf{relname}$ are disjoint.

- Function $sort : \mathbf{relname} \to \mathcal{P}^{fin}(\mathbf{att})$ associates with each relation name R a finite set of attributes $sort(R)$.

  Proviso: $sort^{-1}(U)$ is infinite, for each $U \in \mathcal{P}^{fin}(\mathbf{att})$.

- $arity(R) = |sort(R)|$ is the number of attributes

- Notation: Often $R[U]$ where $U = sort(R)$, or

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

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

denoted `Account: number, branch, customerId`

- A *relation schema* is a relation name R
- 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

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

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

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

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

- $U = \emptyset$: Empty tuple, denoted $\langle\,\rangle$.

- Notation:

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

  $t[V]$ … restriction of t to $V \subseteq U$

Unnamed Perspective

- Other view: Ignore names of attributes, only arity of relations is available
- View a tuple as element of a Cartesian product of $\mathbf{dom}$.
- A tuple t of arity $n \geq 0$ is an element of $\mathbf{dom}^n$.

  Example: tuple t = $\langle \mathtt{Shining}, \mathtt{Kubrick}, \mathtt{Nicholson} \rangle$
- Access components via position $i \in \{1, \ldots, n\}$:

  $t(2)$ = `Kubrick`
- Note: Because of $\leq_{\mathbf{att}}$, unnamed and named perspective naturally correspond

Relation and Database Instances

- 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 $\mathbf{I}$ which assigns each $R \in \mathbf{R}$ a relation instance.
- Other perspectives: Logic programming, first-order logic

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

- Reconstruct a database instance $\mathbf{I}$ as *extended relational theory* $\Sigma_{\mathbf{I}}$:
  - 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 same arity.
  - Unique Name axioms: $\neg(c_i = c_j)$ for each pair 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 **I**.
- if '=' is not available, its intended meaning can be emulated with equality axioms.
- The interpretations of $\mathbf{dom}$, $\mathbf{R}$ satisfying $\Sigma_{\mathbf{I}}$ are isomorphic to $\mathbf{I}$
- 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

- Find titles of current movies:

answer	title
	Shining
	Player
	Chinatown
	Repulsion

- Find theaters showing movies directed by Polanski:

answer	theater
	Le Champo
	Odéon

- Find theaters showing movies featuring Nicholson:

answer	theater
	Le Champo
	Odéon

- Find all directors who acted themselves:

answer	director
	Polanski

- Find directors whose movies are playing in all theaters:

answer	director
	Polanski

- Find theaters that only show 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: Tables constructed from tables 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.
- Database system figures out **how** to get the result, and gives it to the user.
- Database system operates internally with different, **procedural** languages, which specify how to get the result.

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 movies do

  output t

end
```

Declarative vs Procedural

- Theoretical languages:

  Declarative: relational calculus, rule-based queries

  Procedural: relational algebra

- Practical languages: mix of both, but mostly one uses declarative features. (E.g., SQL / Embedded SQL)

Conjunctive Queries

- Simple form of declarative, *rule-base queries*
- A *rule* says *when* certain elements belong to the answer.
- Find titles of current movies:

answer(tl) :– movie(tl, dir, act)

- That is, while (tl, dir, act) ranges over relation movies, output tl (the title attribute)

Next example

- Find theaters showing movies directed by Polanski:

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.

- Queries like this and the previous one are called *conjunctive queries*;

Next example

- Find theaters 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) corresponding to the title tl in relation movie, and output th.

This is the most common type of queries one asks.

- Find directors who acted in 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.

Formal Definition

A *rule-based conjunctive query with (in)equality* q is an expression of form

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

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 must occur in some atom $R_i(\vec{x_i})$ where $R_i \in \mathbf{R}$

Note: Equality '=' can be often eliminated

Semantics

The *result* (aka image) of a conjunctive query q of form (1) on database instance $\mathbf{I}$ is

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

where a valuation ν over $var(q)$ is a mapping

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

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

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)

- $q(\mathbf{I})$ is finite, for any database instance $\mathbf{I}$.
- q is monotonic, ie., $\mathbf{I} \subseteq$ **J** implies $q(\mathbf{I}) \subseteq q(\mathbf{J})$, for every database instances $\mathbf{I}$ and **J**
- q is satisfiable, ie., there exists some **I** such that $q(\mathbf{I}) \neq \emptyset$, provided that '=', '$\neq$' does not occur in q.

A more complicated example

- Find directors whose movies are playing in all theaters.
- "All" is often problematic: one needs *universal quantifier* $\forall$.
- We use notation from mathematical logic:
- $\{$ dir $\mid$ $\forall$ (th, tl) $\in$ schedule
  $\exists$ (tl', act): (tl',dir,act) $\in$ movie $\wedge$ (th, tl') $\in$ schedule $\}$
- 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

Reminder:

- $\forall$ means "for all", $\exists$ means "exists"
- $\wedge$ is conjunction (logical AND)

Can we formulate this as a conjunctive query ?

SQL

- Structured Query Language (declarative)
- Latest standard: SQL-99, or SQL3, well over 1,000 pages
- ”The nice thing about standards is that you have so many to choose from.”
  – Andrew S. Tanenbaum.
- De-facto standard of the relational DB world – replaced all other languages.

Query structure:	`SELECT`	attribute list $\langle R_i.A_j \rangle$
	`FROM`	$R_1, \ldots, R_n$
	`WHERE`	condition c

Simple c: a conjunction of equalities/inequalities

SQL examples

- Find theaters showing movies directed by Polanski:

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

- list directors and theaters in which their movies are playing

```
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, and by 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 ρ of attributes.

Projection

- Chooses some attributes in a relation
- $\pi_{A_1,\ldots,A_n}(R)$: only leaves attributes $A_1, \ldots, A_n$ in relation R.
- Example:

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

- Provides the user with a *view* of data by hiding some attributes

Selection

- Chooses tuples that satisfy some condition
- $\sigma_c(R)$: only leaves tuples t for which $c(t)$ is true
- Conditions: conjunctions of

  $R.A = R.A'$ – two attributes are equal

  $R.A = constant$ – the value of an attribute is a given constant

  Same as above but with $\neq$ instead of $=$
- Examples:

  Movie.Actor=Movie.Director

  Movie.Actor $\neq$ 'Nicholson'

  Movie.Actor=Movie.Director $\wedge$ Movie.Actor='Nicholson'
- Provides the user with a *view* of data by hiding tuples that do not satisfy the condition the user wants.

Selection: Example

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

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

Cartesian Product

- Puts together two relations
- $R_1 \times R_2$ puts together each tuple t_1 of R_1 and each tuple t_2 of 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 renamed attributes to include the name of the relation: in the resulting table, all attributes must have different names.

Cartesian Product: Example

Find theaters 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 $R_2 = \sigma_{cond}(R_1)$ where *cond* is Movie.title = Schedule.title
- We are only interested in movies directed by Polanski, so

  $R_3 = \sigma_{\mathsf{director='Polanski'}}(R_2)$
- In the output, we only want theaters, so finally

  Answer = $\pi_{\mathsf{theater}}(R_3)$

- Summing up, the answer is

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

- Merging selections, this is equivalent to

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

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:

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

- 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 a convenient method 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.
- Example (again): Find theaters playing movies directed by Polanski:

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

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

- SPC Algebra is often identified with the unnamed setting
- For other fragments of Relational Algebra, named perspective is essential (to be seen later)
- Mixed use of settings can be convenient

SQL and Relational Algebra

- We have to translate declarative languages into procedural languages
- Idea:

  `SELECT` is projection π

  `FROM` is Cartesian product $\times$

  `WHERE` is selection σ

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

`SELECT`	$A, B, \cdots$
`FROM`	R
`WHERE`	condition c

is translated into

$$\pi_{A,B,\cdots}\Big(\sigma_c(R)\Big)$$

Translating declarative queries into relational algebra

- Find titles of all movies
- answer(tl) :– movie(tl,dir,act)
- ```
  SELECT Title
  FROM Movie
  ```
- This is simply projection:

$$\pi_{\mathsf{title}}(\text{Movie})$$
  ```

Translation Examples

- Find theaters 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 into a rule such that:

  constants appear only in conditions

  no two variables are the same
- This gives us:

  answer(th) :– schedule(th,tl), movie(tl',dir,act), dir = 'Polanski', tl=tl'
  ```

Translation Examples cont'd

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

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

Translation Examples cont'd

Summing up, the answer is:

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

or, using the rule $\sigma_{c_1}(\sigma_{c_2}(R)) = \sigma_{c_1 \wedge c_2}(R)$:

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

Formal translation: SQL to rule-based queries

`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

Rules into Relational Algebra

- How are rules translated into algebra?

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

Wlog,

$R_1, \ldots R_k \in \mathbf{R}$, $k \leq n$, $R_{k+1}, \ldots, R_n \in \{=, \neq\}$;

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

- First, make sure that no variable occurs in $R_1(\vec{x}_1)$, ..., $R_k(\vec{x}_k)$, at most once:

  If we have $R_i(\ldots, x, \ldots)$ and $R_j(\ldots, x, \ldots)$, turn them into $R_i(\ldots, x', \ldots)$ and $R_j(\ldots, x'', \ldots)$, add $x' = x''$ to the conditions, and if x occurs elsewhere, also $x = x'$

- For 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"

- Replace each occurrence of a constant a in an atom $R_i(..., a, ...)$, $R_i \in \mathbf{R}$, by some variable X and add $X = a$ to the conditions

- Rewritten such rules of form (2) are translated into

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

where $\widehat{\alpha}$ maps each variable x in α to the corresponding attribute in $sort(R_i)$ such that we have $R_i(..., x, ...)$, $R_i \in \mathbf{R}$, in the rule

Putting it together: SQL into relational algebra

- Combining translations:

  SQL into rule-based queries and rule-based into relational algebra

- We have the following SQL to relational algebra translation:

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

  is translated into

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

Another example

- Find theaters showing 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'
  ```

Another example cont'd

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:

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

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

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

- Step 4: we need only theaters in the output

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

Summing up:

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

SPC Algebra into SQL

- Converse mapping feasible as well
- Different ways to show this
- Direct Proof: useful normal form for SPC algebra

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

“simple SPC queries”

- Easy mapping to SQL
- Indirect: Equivalence to query language which is equivalent to SQL

Extension: Natural Join

- Combine all pairs of tuples t_1 and t_2 in relations R_1 resp. R_2 that match on joint 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
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

Join cont'd

- Join is not a new operation of relational algebra
- It is **definable** with $\pi, \sigma, \times$
- Suppose
  - R is a relation with attributes $A_1, \ldots, A_n,\ B_1, \ldots, B_k$
  - S is a relation with attributes $A_1, \ldots, A_n,\ C_1, \ldots, C_m$
  - $R \bowtie S$ has attributes $A_1, \ldots, A_n,\ B_1, \ldots, B_k, C_1, \ldots, C_m$

$$R \bowtie S =$$

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

- Note: named perspective is crucial

Select-Project-Join (SPJ) Queries

Queries of form

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

- These are the most common queries, correspond to simple rules.
- Example: Find theaters showing movies directed by Polanski:
- answer(th) :– schedule(th,tl), movie(tl,'Polanski',act)
- As SPJ query:

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

- What is simpler compared to earlier version?

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

- Selection Schedule.title=Movie.title is eliminated; it is implied by the join.

SPJ queries cont'd

- Find theaters showing movies featuring Nicholson.
- answer(th) :– movie(tl, dir, 'Nicholson'), schedule(th, tl)
- As SPJ query:

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

Translating SPJ queries back into rules and SQL

- $Q = \pi_{A_1,\ldots,A_n}(\sigma_c(R \bowtie S))$
- 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 $A_1 \ldots, A_k$ resp. $C_1, \ldots, C_l$)

  answer($A_1, \ldots, A_n$) :– $R(A_1, \ldots, A_k)$, $S(C_1, \ldots, C_l)$,
  $R.B_1 = S.B_1, \ldots, R.B_m = S.B_m, c$

SPJ to SQL: Example

- Find directors of currently playing movies featuring Ford:
- $\pi_{\text{director}}(\sigma_{\text{actor}='\mathrm{Ford}'}(\text{Movie} \bowtie \text{Schedule}))$
- In SQL:

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

What we've seen so far

- Simple queries given by SQL `SELECT-FROM-WHERE`
- Same queries are defined by rules
- They are also the same queries as those definable by π, σ, $\times$ in relational algebra, i.e., by SPC queries
- Question: What about SPJ?

  SPJ queries are *not* a normal form for the σ, π, $\bowtie$ - fragment of Relational Algebra

  Need renaming for preventing unwanted joins
- 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:

- Using Renaming, Cartesian Product can be easily emulated.
- Also SQL provides renaming construct

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

```
SELECT Father AS Parent, Child
FROM R1
```

Nested SQL queries: simple example

- So far in the `WHERE` clause we used comparisons of attributes.
- In general, a `WHERE` clause could contain *another query*, and test some relationship between an attribute and the result of that query.
- We call queries like this *nested*, as they use *subqueries*
- Example: Find theaters showing Polanski's movies

```
SELECT Schedule.Theater
FROM Schedule
WHERE Schedule.Title IN
         (SELECT Movie.Title
          FROM Movie
          WHERE Movie.Director='Polanski')
```

Nested queries: comparison

```
SELECT S.Theater
FROM Schedule S
WHERE S.Title IN
   (SELECT M.Title
    FROM Movie M
    WHERE M.Director='Polanski')
```

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

- These express the same query
- On the left, each subquery refers to one relation
- The real advantage of nesting is that one can use more complex predicates than `IN`.

Equivalence Theorem

Theorem.

SPJR Queries

= SPC Queries

= simple SPC queries

= conjunctive queries

= SQL `SELECT-FROM-WHERE`

= SQL `SELECT-FROM-WHERE` with `IN`-nesting

Disjunction in queries

- Find actors who 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?
- NO!
  ```

Disjunction in queries cont'd

- Solution: Disjunction can be represented by 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 another syntax for that:

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

Disjunction in queries cont'd

- How to translate a query with disjunction into relational algebra?
- answer(act) :– movie(tl,dir,act), dir='Kubrick'

  is translated into

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

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

  is translated into

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

- The whole query is translated into $Q_1 \cup Q_2$

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

Union in relational algebra

- Another operation of relational algebra: union
- $R \cup S$ is the union of relations R and S
- R and S must have the same set of attributes.
- 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 SPCU-Algebra, or *positive relational algebra*.

Interaction of relational algebra operators

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

SPCU queries

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

Proof: propagate the union operation.

Example:

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

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

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

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

Equivalences II

Theorem.

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 ($\sigma, \pi, \bowtie, \rho, \cup$)

Question: is INTERSECTION an SPJRU query?

That is, given R, S with the same set of attributes, find $R \cap S$.

Bibliography

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Rockville, Md., 1983.

[3] J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.