Database Theory: Lecture 1

Introduction and the relational model

Dr G.M. Bierman

www.cl.cam.ac.uk/Teaching/current/DBaseThy

Background

A **database model** contains the means for

- Specifying particular data structures
- Constraining the data sets
- Manipulating the data

A **DDL** (Data Definition Language) provides the means to specify the structures and constraints.

A **DML** (Data Manipulation Language) provides the means to manipulate the data; specifically querying and updating.

Database models

Database models are typically characterized by the prominent data structure, e.g.

- Graphs:
  - Network model
  - Semantic model
  - Object model
  - XML (?)
- Trees:
  - Hierarchical model
- Relations:
  - Relational model

This lecture course

We will study some of these database models from a *theoretical* perspective.

- Study closely the data model
- Study closely the associated query languages:
  - Choices of query languages
  - Relationships between them
  - Their characteristics (typically expressivity)

Three database models considered in this course:

1. Relational
2. Object
3. Semi-structured/XML

Warning

This is still a course under development!

There will be **eight** lectures, **two** examples classes, and **two** lecturers!

Schedule:

1. GMB - (now!) relational model
2. AD - Relational calculus
3. AD - Deductive databases
4. AD - Recursion and negation
5. AD - Expressivity and complexity
6. GMB - Complex values
7. GMB - Object model
8. GMB - Semistructured/XML model

Books

Recommended textbooks:

- **Foundations of databases** Abiteboul, Hull and Vianu. Addison Wesley, 1995
- **Data on the web** Abiteboul, Buneman and Suciu. Morgan Kaufmann, 2000.

Also: Some useful information in any of the database books from the IB Databases course.

The relational model

What is the relational model?

- Codd 70:
  - Data structure: flat relations
  - Simple algebra of queries
  - No constraints
  - No updates
- Codd 72-:
  - Same data structure
  - Second query language (based on FOL)
  - Proof of equivalence to algebra language
  - Integrity constraints - functional dependencies

What is the relational model? cont.

Therafter a whole host of extensions. Thus the relational model means the **class** of database models that have relations as the core data structure and incorporate some of Codd's approach.

Basics

- Fix a countably infinite set $\mathbf{att}$ of attribute names, with a total order $\leq_{\mathbf{att}}$
- Fix a *single* countably infinite set $\mathbf{dom}$: the underlying domain
- Fix a countably infinite set $\mathbf{relname}$ of relation names
- Assume that there is a *sort* for each element of $\mathbf{relname}$, i.e. a function $sort : \mathbf{relname} \rightarrow \mathcal{P}(\mathbf{att})$.
- The *arity* of a relation R is then defined

$$arity(R) = |sort(R)|$$

- A *relation schema* is just a relation name. We may write $R[U]$ to denote R is of sort U, and $R[n]$ to denote it is of arity n
- A *database schema* is just a nonempty finite set of relation schema, e.g. $\mathcal{R} = \{R_1[U_1], ..., R_k[U_k]\}$

Example relations

$$Cinema = \{Movies[3], Location[3], Guide[3]\}$$

Movies	Title	Director	Actor
	Magnolia	Anderson	Moore
	Magnolia	Anderson	Cruise
	Spiderman	Raimi	Maguire
	Spiderman	Raimi	Dunst
	...		
	Rocky	Avildsen	Stallone
	RockyII	Stallone	Stallone

Guide	Title	Cinema	Time
	Rocky	Warner	12:00
	Spiderman	Picturehouse	19:00
	...		
	Spiderman	Phoenix	19:00
	Magnolia	Picturehouse	22:00

Location	Cinema	Address	Tel
	Picturehouse	Cambridge	504444
	Phoenix	Oxford	512526
	Warner	Cambridge	560225

Choicepoint: names

Q: Are the attribute names part of the explicit database schema or not?

In SQL they are available, e.g. `SELECT p.Name FROM Persons p`

But are they compiled away by the system to just integers?

More theoretically:

- **Named**: A tuple over relation schema $R[U]$ is a map from U to $\mathbf{dom}$
- **Unnamed**: A tuple over relation schema $R[n]$ is a element of the cartesian product $\mathbf{dom}^n$

The choice of model impacts on the query language.

Having assumed a total order on the attribute names provides us with a correspondence between the two. Thus we'll flip between the two during this course

Unnamed: SPC algebra

(Simplest possible algebra first)

SPC query	q	$::=$	R	Base relation
		$\mid$	$\{\langle a \rangle\}$	Singleton constant relation
		$\mid$	$\sigma_{j=a}(q)$	Select (constant)
		$\mid$	$\sigma_{j=k}(q)$	Select (attribute)
		$\mid$	$\pi_{j_1,\ldots,j_k}(q)$	Projection
		$\mid$	$q \times q$	Cartesian product

Example

List the name and addresses of cinemas playing a Almodovar film

$$\pi_{2,3}(\sigma_{1=2}(\pi_5(\sigma_{1=4}(\sigma_{2=\text{Almodovar}}(Movies) \times Guide)) \times Location))$$

Alternatively:

$$\pi_{5,8}(\sigma_{5=7}(\sigma_{1=4}(\sigma_{2=\text{Almodovar}}(Movies \times Guide \times Location))))$$

What is this? (Assume R is of arity n)

$$\pi_{1,\ldots,n}(\sigma_{i=n+1}(R \times \{\langle a \rangle\}))$$

Arity judgements for SPC queries

$$\frac{R[n] \in \mathcal{R}}{\mathcal{R} \vdash R\colon n} \qquad \frac{}{\mathcal{R} \vdash \{\langle a \rangle\}\colon 1}$$

$$\frac{\mathcal{R} \vdash q\colon n \quad n \geq j}{\mathcal{R} \vdash \sigma_{j=a}(q)\colon n} \qquad \frac{\mathcal{R} \vdash q\colon n \quad n \geq max(j,k)}{\mathcal{R} \vdash \sigma_{j=k}(q)\colon n}$$

$$\frac{\mathcal{R} \vdash q\colon n \quad n \geq max(j_1,\ldots,j_k)}{\mathcal{R} \vdash \pi_{j_1,\ldots,j_k}(q)\colon k} \qquad \frac{\mathcal{R} \vdash q_1 : n \quad \mathcal{R} \vdash q_2 : m}{\mathcal{R} \vdash q_1 \times q_2 : n+m}$$

A query q is **well-formed** wrt a schema $\mathcal{R}$ if there exists n such that $\mathcal{R} \vdash q\colon n$.

Semantics

$$\begin{array}{rcl}
[\![R]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \mathcal{I}(R) \\
[\![\{\langle a \rangle\}]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{\langle a \rangle\} \\
[\![\sigma_{j=a}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{t \in [\![q]\!]_{\mathcal{I}} \mid t(j) = a\} \\
[\![\sigma_{j=k}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{t \in [\![q]\!]_{\mathcal{I}} \mid t(j) = t(k)\} \\
[\![\pi_{j_1,\ldots,j_k}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{\langle t(j_1),\ldots,t(j_k)\rangle \mid t \in [\![q]\!]_{\mathcal{I}}\} \\
[\![q_1 \times q_2]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{\langle t(1),\ldots,t(n),s(1),\ldots,s(m)\rangle \mid t \in [\![q_1]\!]_{\mathcal{I}}, s \in [\![q_2]\!]_{\mathcal{I}}\} \\
 & & \text{where} \quad \mathcal{R} \vdash q_1 : n \\
 & & \qquad\qquad \mathcal{R} \vdash q_2 : m
\end{array}$$

Aside: Some queries can return $\emptyset$ for all instances, e.g.
$\forall \mathcal{I}, a \neq b, arity(R) \geq 1$

$$[\![\sigma_{1=a}(\sigma_{1=b}(R))]\!]_{\mathcal{I}} = \emptyset$$

Generalized SPC algebra

1. **Intersection** $R_1 \cap R_2$
2. **Generalized Selection**

   Of form $\sigma_F(q)$ where $F = \gamma_1 \wedge \cdots \wedge \gamma_n$ and $n \geq 1$ and γ_i is $j = a$ or $j = k$
3. **Equijoin**

   $q_1 \bowtie_F q_2$ where $F = \gamma_1 \wedge \cdots \wedge \gamma_n$ where γ_i is $j = k$.

$$\frac{\mathcal{R} \vdash q_1 : n_1 \quad \mathcal{R} \vdash q_2 : n_2 \quad n_1 \geq max(lhs(F)) \quad n_2 \geq max(rhs(F))}{\mathcal{R} \vdash q_1 \bowtie_F q_2 : n_1 + n_2}$$

Exercise 1

1. *Code up these operators*
2. *Formalise the semantics of the equijoin operator*

Normal forms

Define an SPC normal form to be a query of the form

$$\pi_{j_1,\ldots,j_n}(\{\langle a_1\rangle\} \times \cdots \times \{\langle a_m\rangle\} \times \sigma_F(R_1 \times \cdots \times R_k))$$

where $n, m, k \geq 0, \{1, \ldots, m\} \subseteq \{j_1, \ldots, j_n\}$

Proposition 1 *For every (extended) SPC query q, there exists an extended query q_{nf} in normal form such that $[\![q]\!] = [\![q_{nf}]\!]$.*

Named perspective

- Recall: Tuples are functions U to $\mathbf{dom}$
- We write them as, e.g. $\langle A\colon a, B\colon b\rangle$
- Note: We do *not* allow repeated names in a relation sort
- We have to ensure that the result of a query does not have repeated names, e.g. $R \times R$
  - We'll need to be able to rename attributes
- A *natural* join operator reveals itself: join on the common attributes

Named: SPJR algebra

SPJR query	q	$::=$	R	Base relation
		$\mid$	$\{\langle A\colon a\rangle\}$	Singleton constant relation
		$\mid$	$\sigma_{A=a}(q)$	Select (constant)
		$\mid$	$\sigma_{A=B}(q)$	Select (attributes)
		$\mid$	$\pi_{A_1,\ldots,A_k}(q)$	Projection
		$\mid$	$\delta_f(q)$	Renaming
		$\mid$	$q \bowtie q$	Natural join

Sort judgements for SPJR queries

$$\frac{R[U] \in \mathcal{R}}{\mathcal{R} \vdash R\colon U} \qquad \frac{}{\mathcal{R} \vdash \{\langle A\colon a\rangle\}\colon A}$$

$$\frac{\mathcal{R} \vdash q\colon U \quad A \in U}{\mathcal{R} \vdash \sigma_{A=a}(q)\colon U} \qquad \frac{\mathcal{R} \vdash q\colon U \quad A, B \in U}{\mathcal{R} \vdash \sigma_{A=B}(q)\colon U}$$

$$\frac{\mathcal{R} \vdash q\colon U \quad \{A_1, \ldots, A_n\} \subseteq U \quad A_i \text{ distinct}}{\mathcal{R} \vdash \pi_{A_1,\ldots,A_n}(q)\colon A_1, \ldots, A_n}$$

$$\frac{\mathcal{R} \vdash q_1\colon U_1 \quad \mathcal{R} \vdash q_2\colon U_2}{\mathcal{R} \vdash q_1 \bowtie q_2\colon U_1 \cup U_2} \qquad \frac{\mathcal{R} \vdash q\colon U}{\mathcal{R} \vdash \delta_f(q)\colon f(U)}$$

Natural join

$$\frac{\mathcal{R} \vdash q_1\colon U_1 \quad \mathcal{R} \vdash q_2\colon U_2}{\mathcal{R} \vdash q_1 \bowtie q_2\colon U_1 \cup U_2}$$

- If $U_1 \cap U_2 = \emptyset$ then $\bowtie = \times$
- If $U_1 \cap U_2 = U_1 = U_2$ then $\bowtie = \cap$

More

- Can define generalized selection (as for SPC algebra)
- Normal form:

$$\pi_{B_1,\ldots,B_n}(\{\langle A_1: a_1\rangle\} \bowtie \cdots \bowtie \{\langle A_m: a_m\rangle\} \bowtie \sigma_F(\delta_{f_1}(R_1) \bowtie \cdots \bowtie \delta_{f_k}(R_k)))$$

where $n, m \geq 0, \{A_1, ..., A_m\} \subseteq \{B_1, ..., B_n\}$ and A_i are distinct

- A corresponding normal form theorem

Union

We'd like to add some disjunctive flavour to our query language, e.g.

Where can I watch "Magnolia" or "Spiderman"?

1. **Union**

$$\pi_{Cinema}(\sigma_{Title=\text{Magnolia}}(Guide) \cup \sigma_{Title=\text{Spiderman}}(Guide))$$

2. **Extended (disjunctive) selection**

$$\pi_{Cinema}(\sigma_{Title=\text{Magnolia} \vee Title=\text{Spiderman}}(Guide))$$

3. **Non-singleton constant relations**

$$\pi_{Cinema}(Guide \bowtie \{\langle\langle Title\colon \text{Magnolia}\rangle, \langle Title\colon \text{Spiderman}\rangle\rangle\})$$

The addition of Union to our algebras, leads to SPCU and SPJRU algebra

Aside The addition of union to the corresponding calculus is non-trivial!

Further power

Even with Union, our algebra is quite weak, e.g.

Which movies where Stallone did not act were directed by Stallone?

Put more formally, our query algebras are **monotonic**.

Definition 1 *An algebra* A *is* **monotonic** *if* $\forall q \in A.$ *If* $\mathcal{I} \subset \mathcal{I}'$ *then* $[\![q]\!]_{\mathcal{I}} \subseteq [\![q]\!]_{\mathcal{I}'}$.

To rectify this we'll add the **difference** operator.

Difference operator

Unnamed	**Named**
$\dfrac{\mathcal{R} \vdash q_1 : n \quad \mathcal{R} \vdash q_2 : n}{\mathcal{R} \vdash q_1 - q_2 : n}$	$\dfrac{\mathcal{R} \vdash q_1 : U \quad \mathcal{R} \vdash q_2 : U}{\mathcal{R} \vdash q_1 - q_2 : U}$

Our example becomes, e.g.

$$\pi_{Title}(\sigma_{Director=\text{Stallone}}(Movies)) - \pi_{Title}(\sigma_{Actor=\text{Stallone}}(Movies))$$

- The **(unnamed) relational algebra** is SPCU with difference operator.
- The **(named) relational algebra** is SPJRU with difference operator.

(It is traditional to add to these the generalizations, e.g. $\cap$ for unnamed relational algebra)

What's coming up

- **Relational model**
  - Different query language perspective: **logic**
  - Obvious questions:
    - Equivalence with algebra
    - Expressivity?
    - Natural extensions and their expressivity
- **Different data models**

Database Theory: Lecture 2

The Relational Calculus

Dr A. Dawar

www.cl.cam.ac.uk/Teaching/current/DBaseThy

Queries

A **query** is a formal syntactic rendering of a question we wish to pose to a database.

Examples

- Who is the director of "Spiderman"?
- Where is "Spiderman" playing?
- Is there a film directed by Almodovar playing in Cambridge?

Formally, a query q defines a **mapping**. It has an **input schema** (typically a database schema) and an **output schema** (typically a relation schema).

The mapping takes an instance $\mathcal{I}$ over the input schema to an instance $[\![q]\!]_{\mathcal{I}}$ over the output schema.

Queries in Algebra

In the (SPC or SPJR) relational algebra, a query is a term in the algebra.

Each term denotes a relation.

Relations are combined using various unary operations (*selection*, *projection*) and binary operations (*Cartesian product*, *union*).

A **Boolean** query is one where the output schema is a relation of arity 0.

So, for any instance $\mathcal{I}$, $[\![q]\!]_{\mathcal{I}}$ is either $\{\}$ or $\{\langle\rangle\}$.

Conjunctive Queries

The various **relational calculi** (of which the **conjunctive calculus** is the simplest) are based on first-order predicate logic.

If R is a relation of arity 2 and $a, b \in \mathbf{dom}$, then $R(a, b)$ is a **fact** that is either true or false in a given database instance $\mathcal{I}$.

If x is a variable, then $R(a, x)$ is true or false given an instance $\mathcal{I}$ and a **valuation** for x.

$\{x \mid R(a, x)\}$ is a simple query.

Example

- Who is the director of "Spiderman"?

$$\{x \mid \exists z \mathit{Movies}(\text{“Spiderman”}, x, z)\}$$

Conjunctive Calculus

Formally, a **conjunctive formula** ϕ is given by the following syntax

ϕ	$::=$	$R(t_1, \ldots, t_n)$	atom
	$\mid$	$\phi \wedge \phi$	conjunction
	$\mid$	$\exists x\, \phi$	existential quantification

where R is a relation name, x is a variable and each t_i is either a variable or a constant in $\mathbf{dom}$.

A **conjunctive query** has the form

$$\{\langle t_1, \ldots, t_n\rangle \mid \phi\}$$

where

- ϕ is a conjunctive formula;
- each t_i is either a variable or a constant; and
- the set of variables in $t_1, \ldots, t_n$ is exactly $\mathit{free}(\phi)$.

Semantics

A **valuation** is a function $\nu : V \to \mathbf{dom}$, where V is a set of variables.

If V includes all free variables in a formula ϕ, we can define satisfaction $\mathcal{I} \models_\nu \phi$ inductively by:

$$\begin{array}{rll} \phi = R(\bar{t}) & \text{and} & \nu(\bar{t}) \in \mathcal{I}(R) \\ \phi = \phi_1 \wedge \phi_2 & \text{and} & \mathcal{I} \models_\nu \phi_1 \text{ and } \mathcal{I} \models_\nu \phi_2 \\ \phi = \exists x\, \psi & \text{and} & \text{there is a } c \in \mathbf{dom} \text{ with } \mathcal{I} \models_{\nu[x/c]} \psi \end{array}$$

The query mapping q defined by the query $\{\bar{t} \mid \phi\}$ maps the instance $\mathcal{I}$ to

$$[\![q]\!]_\mathcal{I} = \{\nu(\bar{t}) \mid \mathcal{I} \models_\nu \phi \text{ for some } \nu\}$$

Active Domain

The **active domain** of a query q, denoted $adom(q)$ is the set of constants appearing in q.

Similarly, the **active domain** of an instance $\mathcal{I}$, denoted $adom(\mathcal{I})$ is the set of all constants appearing in any relation of $\mathcal{I}$.

Show, by induction, that if $\mathcal{I} \models_\nu \phi$, then for any free variable x of ϕ,

$$\nu(x) \in adom(\mathcal{I}).$$

Exercise!

It follows that in evaluating a query q, we only need to consider valuations whose range is in $adom(q) \cup adom(\mathcal{I})$.

Normal Form

A conjuctive formula is said to be in **normal form** if all quantifiers precede all conjunctions. That is, it's of the form

$$\exists x_1, \ldots, x_m (R_1(t_1) \wedge \cdots \wedge R_n(t_n)).$$

Every query in the conjunctive calculus can be defined with a formula in normal form.

Use,

1. $\exists x\, \phi$ is equivalent to $\exists y\, \phi[x/y]$ if y does not occur in ϕ.
2. $(\exists x\, \phi) \wedge (\exists y\, \psi)$ is equivalent to $\exists x \exists y (\phi \wedge \psi)$ if x is not free in ψ, y is not free in ϕ and $x \neq y$.

Equality

Every query in the calculus defined so far is **satisfiable**.

That is, for every q, there is an $\mathcal{I}$ such that $[\![q]\!]_\mathcal{I}$ is not empty.

Compare $\sigma_{1=a}(\sigma_{1=b}(R))$

The calculus could be extended to express such queries by allowing equalities $(t_1 = t_2)$ as atomic formulae.

$$\{x \mid R(x) \wedge x = a \wedge x = b\}$$

Equivalence

Every query definable in SPC algebra is definable in the conjunctive calculus

Moreover, if the query is satisfiable, then it is definable without the use of equality.

Every query definable in the conjunctive calculus is definable in the SPC algebra

Algebra to Calculus

Given a query q in the SPC algebra, we define a query q^* in the calculus.

$$R^* \stackrel{\text{def}}{=} \{x_1, \ldots, x_n \mid R(x_1, \ldots, x_n)\}$$
$$\{\langle a \rangle\}^* \stackrel{\text{def}}{=} \{a \mid \texttt{true}\}$$

If $q_1^* = \{t_1, \ldots, t_n \mid \phi_1\}$ and $q_2^* = \{u_1, \ldots, u_m \mid \phi_2\}$ and there are no variables in common, then

$$(q_1 \times q_2)^* \stackrel{\text{def}}{=} \{t_1, \ldots, t_n, u_1, \ldots, u_m \mid \phi_1 \wedge \phi_2\}$$
$$(\pi_{j_1,\ldots,j_k}(q_1))^* \stackrel{\text{def}}{=} \{t_{j_1}, \ldots, t_{j_k} \mid \exists y_1, \ldots, y_l\, \phi_1\}$$

where $y_1, \ldots, y_l$ enumerates all variables among $t_1, \ldots, t_n$ that are not in $t_{j_1}, \ldots, t_{j_k}$.

Selection

If $q^* = \{t_1, \ldots, t_n \mid \phi\}$,

$$(\sigma_{j=a}(q))^*$$

is obtained by replacing t_j by a.

For

$$(\sigma_{j=k}(q))^*$$

if either t_j or t_k is a variable, it can be replaced by the other. If they are both constants, we need equality!

Calculus to Algebra

Given a query

$$q = \{\bar{t} \mid \exists \bar{x}(R_1(u_1) \wedge \cdots \wedge R_n(u_n))\}$$

take the cross-product $R_1 \times \cdots \times R_n$,

apply a selection $\sigma_{j=a}$ for each constant a appearing among the u_i,

apply a selection $\sigma_{j=k}$ for every repeated variable among the u_i,

apply a projection $\pi_{j_1,\ldots,j_k}$ to get rid of the columns corresponding to $\bar{x}$,

apply further selections corresponding to constants and repeated variables in $\bar{t}$.

Disjunction

To extend the conjunctive calculus to one equivalent to SPCU algebra, it would appear we need disjunction.

Where can I watch "Magnolia" or "Spiderman"?

$$\{x \mid \exists y\,(\textit{Guide}(\text{“Magnolia”}, x, y) \lor \textit{Guide}(\text{“Spiderman”}, x, y))\}$$

Aside: How would you prove this cannot be done in the conjunctive calculus?

However, allowing unrestricted disjunction can result in **unsafe** queries, i.e. queries with possibly infinite answers.

$$\{x, y, z \mid R(x, y) \lor R(y, z)\}$$

It is no longer the case that a satisfying valuation is restricted to the active domain.

Negation

The full **relational calculus** is obtained by allowing **negation** in addition to disjunction. This allows the full power of first-order predicate logic in specifying a query.

$$q = \{t_1, \ldots, t_n \mid \phi\}$$

with rules for formulae (in addition to those of conjunctive formulae):

ϕ	$::=$	$\neg\phi$	negation
	$\mid$	$\phi \lor \phi$	disjunction
	$\mid$	$\forall x\,\phi$	universal quantification

In the presence of negation, the last two are not strictly necessary, but they are convenient.

Safety

Using negation and disjunction, one can write queries that admit infinite relations as answers.

$$\{x \mid \neg\textit{Movies}(\text{“Kika”}, \text{“Almodovar”}, x)\}$$

$$\{x, y \mid \textit{Movies}(\text{“Kika”}, \text{“Almodovar”}, x) \lor \textit{Movies}(y, \text{“Almodovar”}, \text{“Maura”})\}$$

One possibility might be to restrict $\mathbf{dom}$ to some fixed finite set. Still, some queries are not **domain independent**.

$$\{x \mid \forall y\, R(x, y)\}$$

Domain Independence

If we fix a domain $\mathbf{d} \subseteq \mathbf{dom}$, the interpretation of a query q over an instance $\mathcal{I}$ can be **relativized** to $\mathbf{d}$.

All valuations are restricted to have range in $\mathbf{d}$.

The resulting interpretation is $[\![q]\!]_{\mathbf{d},\mathcal{I}}$.

*A query q is **domain independent** if for any $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$ and any $\mathcal{I}$,*

$$[\![q]\!]_{\mathbf{d},\mathcal{I}} = [\![q]\!]_{\mathbf{d}',\mathcal{I}}.$$

It is undecidable whether a given expression of the relational calculus defines a domain independent query.

If a query is domain independent, it can be evaluated by restricting all valuations to the **active domain**.

Equivalence

Every query expressible in the relational algebra is equivalent to a **domain independent** query in the relational calculus.

Every query q in the relational calculus interpreted with domain restricted to $adom(q, \mathcal{I})$ is equivalent to a query in the relational algebra.

Database Theory: Lecture 3

Deductive Databases

Dr A. Dawar

www.cl.cam.ac.uk/Teaching/current/DBaseThy

Rules

Rules provide an alternative syntax (inspired by logic programming) for **conjunctive queries**.

$$A(x, y) \leftarrow \textit{Movies}(z_1, \text{“Almodovar”}, z_2), \textit{Guide}(z_1, x, z_3), \textit{Location}(x, y, z_4)$$

expresses the query

$$\{x, y \mid \exists z_1, \ldots, z_4\ \textit{Movies}(z_1, \text{“Almodovar”}, z_2) \wedge \textit{Guide}(z_1, x, z_3) \wedge \textit{Location}(x, y, z_4)\}$$

and, in addition, gives the resulting relation the name A.

Syntax

Formally, a **conjunctive rule** is:

$$R(t) \leftarrow R_1(t_1), \ldots, R_n(t_n)$$

where

- R is a relation name **not** in the input database schema.
- Each R_i is a relation in the input schema.
- Each of t and t_i is a tuple of terms (i.e. variables or constants).
- Every variable occurring in t occurs in at least one of the t_i.

Semantics

The meaning of the rule

$$R(t) \leftarrow R_1(t_1), \ldots, R_n(t_n)$$

is given by

$$\{t \mid \exists \bar{x}\, R_1(t_1) \wedge \ldots \wedge R_n(t_n)\}$$

where $\bar{x}$ is the sequence of all variables in $t_1, \ldots, t_n$ not occurring in t.

$R(t)$ is called the **head** of the rule.

$R_1(t_1), \ldots, R_n(t_n)$ is the **body** of the rule.

Aside: by allowing equalities ($x = y$) in the body, we can ensure that there are no constants or repeated variables in the head.

Disjunction

Disjunction is easily added by allowing multiple rules for the same relation

Where can I watch "Magnolia" or "Spiderman"?

$$A(x) \leftarrow \textit{Guide}(\text{"Magnolia"}, x, y)$$

$$A(x) \leftarrow \textit{Guide}(\text{"Spiderman"}, x, y)$$

Note: Occurrences of a variable in distinct rules are to be treated as distinct variables.

This form of disjunction is always **safe**.

Multiple Relations

As the syntax of conjunctive rules allows us to name the result of the query, it is easy to simultaneously define **multiple** relations.

$$\begin{aligned} S_1(x, z) &\leftarrow Q(x, y), R(y, z, w) \\ S_2(x, y, z) &\leftarrow S_1(x, w), R(w, y, v), S_1(v, z) \end{aligned}$$

As long as the dependencies among the relations are **non-circular**, each individual relation is still a conjunctive query. For instance,

$$S_2(x, y, z) \leftarrow Q(x, y_1), R(y_1, w, w_1), R(w, y, v), Q(v, y_2), R(y_2, z, w_2).$$

Relations (such as Q, R) occurring in the database schema are called **extensional database relations**. Relations that occur in the heads of rules are **intensional database relations**.

Negation

One possible extension would be to allow negation on atomic formulae in the body of a rule.

This can allow **unsafe** queries.

$$A(x) \leftarrow \neg \textit{Movies}(\text{"Kika"}, \text{"Almodovar"}, x)$$

However, there is a simple syntactic restriction that guarantees **safety**.

Range Restriction

A **Range-restricted** conjunctive rule with negation is

$$A(t) \leftarrow L_1(t_1), \ldots, L_n(t_n)$$

where

- Each L_i is a **literal**, i.e. either R or $\neg R$ for some relation R.
- Every variable occurring in t occurs in at least one of the t_i.
- Every variable in the body appears in some **positive** literal.

A literal is positive if it's of the form $R(t_i)$. It is negative if it's of the form $\neg R(t_i)$.

Universal Quantification

If negation is restricted to **edb** (extensional database) relations, not all relational calculus queries are expressible.

Who are the actors that appeared in every Almodovar film?

$$\{x \mid \forall y[(\exists z \textit{ Movies}(y, \text{“Almodovar”}, z)) \rightarrow \textit{Movies}(y, \text{“Almodovar”}, x)]\}$$

Informally, a set of conjunctive rules with negation only on **edb** relations is equivalent to a formula that is the disjunction of formulae of the form

$$\exists x_1, \ldots, x_n(L_1(t_1) \wedge \cdots L_m(t_m))$$

Simulating the universal quantifier requires placing a negation outside the existential quantifier.

Universal Quantification

Who are the actors that appeared in every Almodovar film?

$$\text{Almo-actor}(z, y) \leftarrow \textit{Movies}(y, \text{“Almodovar”}, z)$$

$$\text{Not-all}(x) \leftarrow \text{Almo-actor}(x, z_1), \text{Almo-actor}(y, z_2), \neg\text{Almo-actor}(x, z_2)$$

$$\text{All}(x) \leftarrow \text{Almo-actor}(x, z), \neg\text{Not-all}(x).$$

Example: Path Finding

A railway database contains the relation

Railway	Service	From	To
	WAGN	Cambridge	Ely
	WAGN	Cambridge	King's Cross
	Central	Ely	Birmingham
	Virgin	Euston	Birmingham
	...		
	GNER	Peterbrough	Newcastle
	Great Western	Paddington	Cardiff

This is dynamically updated, to reflect cancellations and line failures.

Is it possible to get from Cambridge to Glasgow?

Path Finding

In the relational calculus, we can ask:

Which stations are reachable from Cambridge without change?

$$\text{Reach}_0(x) \leftarrow \textit{Railway}(y, \text{“Cambridge”}, x)$$

Which stations are reachable from Cambridge with one change?

$$\text{Reach}_1(x) \leftarrow \textit{Railway}(y_1, \text{“Cambridge”}, z), \textit{Railway}(y_2, z, x)$$

Which stations are reachable from Cambridge with two changes?

$$\text{Reach}_2(x) \leftarrow \textit{Railway}(y_1, \text{“Cambridge”}, z_1), \textit{Railway}(y_2, z_1, z_2), \textit{Railway}(y_3, z_2, x)$$

But, there is no way to ask “which stations are reachable from Cambridge?”.

Recursion

We can allow **recursion**, by allowing a relation name to appear in both the head and body of a rule.

Example:

$$\begin{aligned} T(x,y) &\leftarrow G(x,y) \\ T(x,y) &\leftarrow G(x,z), T(z,y). \end{aligned}$$

Here, G is a relation given in the database. The query defines a relation T that is the transitive closure of G.

Datalog is the language of conjunctive rules with recursion.

Route Finding

Which stations are reachable from Cambridge?

$$\begin{aligned} \text{Reach}(x) &\leftarrow \textit{Railway}(y, \text{“Cambridge”}, x) \\ \text{Reach}(x) &\leftarrow \textit{Railway}(y_1, z, x), \text{Reach}(z) \end{aligned}$$

Is it possible to get from Cambridge to Glasgow?

$$\text{Camb-Glas} \leftarrow \text{Reach}(\text{“Glasgow”})$$

Syntax

A **Datalog rule** is an expression

$$R(t) \leftarrow R_1(t_1), \ldots, R_n(t_n)$$

where

- The tuples of terms $t, t_1, \ldots, t_n$ satisfy the restrictions on conjunctive rules.
- Each R_i is either a relation in the input database scheme, or appears in the head of some rule.

A **Datalog program** P is a sequence of **rules**.

We write $edb(P)$ for the database schema consisting of the relation names that occur only in the body of rules in P. $sch(P)$ is the schema consisting of all relations occurring in P.

Note: no negation.

Model-Theoretic Semantics

With a rule

$$R(t) \leftarrow R_1(t_1), \ldots, R_n(t_n)$$

we associate the formula of first-order predicate logic:

$$\forall x_1, \ldots, x_n \, [(R_1(t_1) \wedge \cdots \wedge R_n(t_n)) \rightarrow R(t)]$$

where $x_1, \ldots, x_n$ enumerates **all** the variables appearing in the rule.

For a program P, we write Σ_P for the **conjunction** of all formulae associated with rules in P.

Since there are no free variables in Σ_P, for any instance $\mathcal{I}$ over $sch(P)$, we have either

$\mathcal{I} \models \Sigma_P$ $\quad$ $\mathcal{I}$ satisfies Σ_P.

or

$\mathcal{I} \not\models \Sigma_P$ $\quad$ $\mathcal{I}$ does not satisfy Σ_P.

Model-Theoretic Semantics *contd.*

If P is a program, a database instance over $sch(P)$ is said to be a **model** of P if it satisfies Σ_P.

If $\mathcal{I}$ is a database instance over $edb(P)$, the semantics of P over $\mathcal{I}$ is defined to be the **minimum** instance $\mathcal{J}$ such that $\mathcal{I} \subseteq \mathcal{J}$ and $\mathcal{J}$ is a model of P.

How do we know such a model exists?

Fixed-point Semantics

Let P be a program and $\mathcal{I}$ an instance over $sch(P)$.

Define the instance $T_P(\mathcal{I})$ as follows

- For every relation R in $edb(P)$, $T_P(\mathcal{I})(R) = \mathcal{I}(R)$.
- For every relation R not in $edb(P)$, a tuple u is in R if, and only if, there is a rule

  $$R(\bar{x}) \leftarrow R_1(t_1), \ldots, R_n(t_n)$$

  and a valuation ν such that $\nu(\bar{x}) = u$ and $\mathcal{I} \models_\nu R_i(t_i)$ for each i.

That is, $T_P(\mathcal{I})$ is formed of the set of facts that can be **immediately justified** by the program P and the instance $\mathcal{I}$.

Fixed-point Semantics *contd.*

The function T_P is a **monotone map** on instances.

If $\mathcal{I} \subseteq \mathcal{J}$, then $T_P(\mathcal{I}) \subseteq T_P(\mathcal{J})$.

Exercise: prove it!

Fixed-point theorem

For any instance $\mathcal{I}$, there is a least $\mathcal{J}$ such that $\mathcal{I} \subseteq \mathcal{J}$ which is a fixed-point of T_P.

$$T_P(\mathcal{J}) = \mathcal{J}.$$

Iteration

The least fixed point can be reached by iteration.

Given a program P and an instance $\mathcal{I}$ over $edb(P)$, take

$\mathcal{J}_0 = \mathcal{I}$

$\mathcal{J}_{i+1} = T_P(\mathcal{J}_i)$.

Repeat until $\mathcal{J}_{i+1} = \mathcal{J}_i$.

Database Theory: Lecture 4

Recursion and Negation

Dr A. Dawar

www.cl.cam.ac.uk/Teaching/current/DBaseThy

Datalog

Datalog extends the **conjunctive calculus** with recursion, but does not allow negation.

We can express

> *Which stations are reachable from Cambridge?*

but not

> *For which pairs of stations is there no direct connection?*

The expressive power of Datalog and the relational calculus are incomparable.

Monotonicity

Every query definable in Datalog is **monotonic**.

If P is a Datalog program and $\mathcal{I}, \mathcal{I}'$ instances over $edb(P)$, then

If $\mathcal{I} \subseteq \mathcal{I}'$ then $P(\mathcal{I}) \subseteq P(\mathcal{I}')$.

$$
\begin{array}{lcl}
\mathcal{I} & \subseteq & \mathcal{I}' \\
T_P(\mathcal{I}) & \subseteq & T_P(\mathcal{I}') \\
T_P(T_P(\mathcal{I})) & \subseteq & T_P(T_P(\mathcal{I}')) \\
T_P^3(\mathcal{I}) & \subseteq & T_P^3(\mathcal{I}') \\
 & \vdots &
\end{array}
$$

Example

Given a graph as a binary relation

Edge	Source	Target
	a	c
	a	b
	b	d

We can ask for which pairs of nodes there is a path:

$$\begin{aligned} \text{Path}(x,y) &\leftarrow \textit{Edge}(x,y) \\ \text{Path}(x,y) &\leftarrow \textit{Edge}(x,z), \text{Path}(z,y). \end{aligned}$$

But we cannot ask: for which pairs x, y is every path of even length.

Adding the pair (a, d) to the above table would require removing it from the result of the query. This shows that the query is not monotone.

Adding Negation

Allowing negation in datalog programs is not a trivial task, as the fixed-point semantics was based on monotonicity.

We now consider a series of ever more expressive query languages obtained by different ways of incorporating negation into datalog.

All have the *proviso* that negation is **range-restricted**.

> That is, if a $\neg R(t)$ appears in the body of a rule, every variable in t appears in some positive literal in the body of the same rule.

Semipositive Datalog

The simplest method of extending datalog is to allow negation only on **edb** predicates.

A **semipositive datalog** program is a collection of rules

$$R(t) \leftarrow L_1(t_1), \ldots, L_n(t_n)$$

where if L_i is $\neg R_i$, then R_i does not appear in the head of any rule.

and the usual restrictions on variables apply.

$$\begin{aligned} T(x,y) &\leftarrow \neg G(x,y), G(x,z), G(y,w) \\ T(x,y) &\leftarrow \neg G(x,z), T(z,y), G(x,w). \end{aligned}$$

This computes the transitive closure of the **complement** of G.

Semipositive Datalog *contd.*

For a semipositive datalog program P, the map T_P on instances is still monotone in a restricted sense.

If $\mathcal{I}$ and $\mathcal{I}'$ are instances over $sch(P)$ such that:

- $\mathcal{I} \subseteq \mathcal{I}'$, and
- for each R in $edb(P), \mathcal{I}(R) = \mathcal{I}'(R)$.

then

$$T_P(\mathcal{I}) \subseteq T_P(\mathcal{I}').$$

This suffices for proving the existence of minimal fixed-points and therefore giving a semantics for semipositive datalog.

Semipositive Datalog *contd.*

While semipositive datalog can express some queries that are neither in the relational calculus nor in datalog, it cannot express universal quantification.

$$\{x \mid \forall y[(\exists z \textit{ Movies}(y, \text{“Almodovar”}, z)) \rightarrow \textit{Movies}(y, \text{“Almodovar”}, x)]\}$$

Suppose $\mathcal{I} \subseteq \mathcal{I}'$ are two instances over $edb(P)$ such that for any tuple u over $adom(\mathcal{I})$, $u \in \mathcal{I}(R)$ if, and only if, $u \in \mathcal{I}'(R)$.

Then $P(\mathcal{I}) \subseteq P(\mathcal{I}')$.

The proof of this is similar to monotonicity of ordinary datalog programs.

To get universal quantification, we have to find a way of alternating rule application with negation.

Stratified Datalog

A **stratified datalog** program P is a set of rules

$$R(t) \leftarrow L_1(t_1), \ldots, L_n(t_n),$$

which can be partitioned into sets $P = P_1 \cup \cdots \cup P_n$ such that

- For every relation R, there is an i such that all rules with R in the head are in P_i (we call i the defining stratum for R; for **edb** relations, the defining stratum is 0).
- If L_j is a positive literal R_j, then the defining stratum of $R_j \leq$ the defining stratum of R.
- If L_j is a negative literal R_j, then the defining stratum of $R_j <$ the defining stratum of R.

Example

$$\begin{array}{rcl} \text{Almo-actor}(z, y) & \leftarrow & \textit{Movies}(y, \text{“Almodovar”}, z) \\ \text{Not-all}(x) & \leftarrow & \text{Almo-actor}(x, z_1), \text{Almo-actor}(y, z_2), \neg\text{Almo-actor}(x, z_2) \\ \text{All}(x) & \leftarrow & \text{Almo-actor}(x, z), \neg\text{Not-all}(x). \end{array}$$

This example does not involve recursion.

Indeed, stratified datalog without recursion is exactly equivalent to the relational calculus.

Semantics

If $\mathcal{I}$ is a database instance over $edb(P)$, where P is a stratified datalog program $P = P_1 \cup \cdots \cup P_n$, we define the following sequence of instances for $0 \leq i \leq n$.

$$\begin{array}{rcl} \mathcal{I}_0 & \stackrel{\text{def}}{=} & \mathcal{I} \\ \mathcal{I}_{i+1} & \stackrel{\text{def}}{=} & \mathcal{I}_i \cup P_{i+1}(\mathcal{I}_i) \end{array}$$

Then, $P(\mathcal{I}) \stackrel{\text{def}}{=} \mathcal{I}_n$.

For a given program P, there may be more than one valid stratification. Show that the semantics does not depend on which stratification is chosen.

Example: Game

$$\mathcal{G}ame = \{Move[3], Win[2]\}$$

Move	Player	Position	Next
	Black	a	b
	White	a	c
	White	b	d
	...		

Win	Position	Player
	d	White
	...	

$$\text{Win}_0(x) \stackrel{\text{def}}{=} \textit{Win}(x, \text{“White”})$$
$$\text{Win}_{i+1}(x) \stackrel{\text{def}}{=} \exists y\ \textit{Move}(\text{“White”}, x, y) \wedge (\forall z\ \textit{Move}(\text{“Black”}, y, z) \longrightarrow \text{Win}_i(z))$$

Each Win_i is definable in stratified datalog but the set of all winning positions for White is not definable.

Fixed-Point Calculus

Relational equations such as

$$\text{Win-pos}(x) \equiv \textit{Win}(x, \text{“White”}) \vee$$
$$\exists y\ \textit{Move}(\text{“White”}, x, y) \wedge (\forall z\ \textit{Move}(\text{“Black”}, y, z) \longrightarrow \text{Win-pos}(z))$$

have solutions.

If

- ϕ is a relational expression over schema $\sigma \cup \{R\}$;
- no occurrence of R in ϕ is in the scope of a negation; and
- $\mathcal{I}$ is an instance over schema σ.

There is a minimum extension of $\mathcal{I}$ satisfying $R \leftrightarrow \phi$.

Fixed-Point Calculus Definition

Predicate	P	$::=$	R	Base relation
		$\mid$	$\mu_R(\phi)$	where R occurs only positively in ϕ
Formula	ϕ	$::=$	$P(\bar{t})$	with matching arities
		$\mid$	$t = t$	
		$\mid$	$\phi \wedge \phi$	
		$\mid$	$\exists x \phi$	
		$\mid$	$\neg \phi$	

For the semantics, interpret $\mu_R(\phi)$ as the least relation satisfying $R \leftrightarrow \phi$.

Every query definable in stratified datalog is expressible in the fixed-point calculus.

Extending the Algebra

One way of extending **relational algebra** to express recursive queries like **transitive closure**

$$T(x, y) \leftarrow G(x, y)$$
$$T(x, y) \leftarrow G(x, z), T(z, y).$$

is to add *while* loops.

$T := G$
while change do
    begin
    $T := T \cup \pi_{1,4}(\sigma_{2=3}(G \times T))$
    end

While Programs

Every query definable in the fixed-point calculus is definable by a *while* program of the form.

$R := E_1$
while change do
  begin
  $R := E_2$
  end

where E_1 and E_2 are terms of the relational algebra.

Moreover, if E_2 is of the form $R \cup \cdots$, the converse holds as well.

Database Theory: Lecture 5
Expressivity and Complexity

Dr A. Dawar

`www.cl.cam.ac.uk/Teaching/current/DBaseThy`

Evaluating a Query

Given a datalog program P, a naïve method of evaluating it on a database instance $\mathcal{I}$ would proceed as follows:

1. $\mathcal{J} := \mathcal{I} \cup \{R/\emptyset \mid R \in idb(P)\}$.
2. For every valuation ν of the variables occurring in P over $adom(P, \mathcal{I})$, if

$$R(\bar{x}) \leftarrow \text{body}$$

   is a rule such that $\mathcal{J} \models_\nu$ body, then

$$\mathcal{J}(R) := \mathcal{J}(R) \cup \{\nu(\bar{x})\}.$$

3. Repeat step (2) until no new tuples are added.

Running Time

Let

- $n = |adom(P, \mathcal{I})|$
- $k =$ the maximum number of distinct variables occurring in any rule in P
- $l =$ the maximum arity of any relation in the head of a rule in P

Then, the naïve evaluation of P on $\mathcal{I}$ takes $O(n^{k+l})$ steps.

- We only need to consider n^k distinct evaluations for any given rule.
- The number of iterations is at most n^l.

For any program P, there is a polynomial p such that P can be evaluated in time $p(|\mathcal{I}|)$ on an instance $\mathcal{I}$.

Running Time

The upper bound on the running time also holds for stratified datalog, as each stratum is evaluated to completion just once.

One can also derive such bounds for relational algebra.

- For any relational algebra query q, there is a polynomial-time algorithm to evaluate it on all instances.
- The degree of the polynomial is bounded by the maximum arity of any sub-query of q.

Many optimization techniques are designed to reduce the arities of intermediate relations.

NP-complete Query

Is there a way to travel around Britain so that I visit every railway station exactly once?

Since the query expressed above is an **NP-complete** property of a database instance, we cannot hope to express it in **datalog**.

Indeed, one can prove that it is not definable.

Is every polynomial-time computable query expressible in **datalog**? In stratified **datalog**? In the fixed-point calculus?

Polynomial-time Queries

Give me the set of elements in the active domain which are at even positions when listed lexicographically.

This query is computable in polynomial time.

Unless the lexicographical order is explicitly given as a relation, it is not definable in any language we have considered.

The query is not **generic**

Generic Queries

Given database instances $\mathcal{I}, \mathcal{J}$ over the same schema, a bijection

$$\beta : \mathbf{dom} \rightarrow \mathbf{dom}$$

is an **isomorphism** from $\mathcal{I}$ to $\mathcal{J}$ if for any relation R and any tuple u:

$$u \in \mathcal{I}(R) \quad \text{iff} \quad \beta(u) \in \mathcal{J}(R).$$

A query mapping q is **generic** if, whenever β is an isomorphism from $\mathcal{I}$ to $\mathcal{J}$, it is also an isomorphism from $q(\mathcal{I})$ to $(\beta q)(\mathcal{J})$.

βq denotes the query obtained from q by replacing every constant a by $\beta(a)$.

Every query language we have considered only defines generic queries.

Order

Let $\leq$ be a total order relation on $\mathbf{dom}$.

(for instance, if $\mathbf{dom}$ is the set of natural numbers)

Allow queries in the fixed-point calculus to use the order.

Predicate	P	$::=$	R	Base relation
		$\mid$	$\mu_R(\phi)$	where R occurs only positively in ϕ
Formula	ϕ	$::=$	$P(\bar{t})$	with matching arities
		$\mid$	$t \leq t$	
		$\mid$	$\phi \land \phi$	
		$\mid$	$\exists x \phi$	
		$\mid$	$\neg \phi$	

Fixed-Point with Order

The fixed-point calculus with order can express non-generic queries:

Is there a pair (a, b) in the relation R with $a < b$?

Immerman and Vardi showed:

A query is expressible in fixed-point calculus with order if, and only if, it is computable in polynomial time.

In the absence of order, there are polynomial-time, **generic** queries that cannot be expressed in the fixed-point calculus.

Is the number of tuples in R even?

Open Question: Is there a language that can express exactly the polynomial-time computable generic queries?

Proving Inexpressivity

Inexpressivity

The transitive closure query:

$$
\begin{aligned}
T(x,y) &\leftarrow G(x,y) \\
T(x,y) &\leftarrow G(x,z), T(z,y).
\end{aligned}
$$

is not definable in the relational calculus, without recursion.

How do we prove this?

We can show that the query:

Is the graph G strongly connected?

is not definable.

If T were definable, this query would be $\forall x \forall y\, T(x,y)$.

Quantifier Rank

For a formula ϕ, we define its **quantifier rank** $qr(\phi)$ by:

1. if ϕ is atomic then $qr(\phi) = 0$,
2. if $\phi = \neg\psi$ then $qr(\phi) = qr(\psi)$,
3. if $\phi = \psi_1 \vee \psi_2$ or $\phi = \psi_1 \wedge \psi_2$ then $qr(\phi) = \max(qr(\psi_1), qr(\psi_2))$.
4. if $\phi = \exists x\, \psi$ or $\phi = \forall x \psi$ then $qr(\phi) = qr(\psi) + 1$

That is, $qr(\phi)$ is the depth of nesting of quantifiers in ϕ.

Proving Inexpressibility

For two instances $\mathcal{I}, \mathcal{J}$, we write $\mathcal{I} \equiv_m \mathcal{J}$ to denote that for any formula ϕ with $qr(\phi) \leq m$ and no free variables:

$$\mathcal{I} \models \phi \quad \text{iff} \quad \mathcal{J} \models \phi.$$

To prove that strong connectedness is not definable in relational calculus, we show that for each m there are graphs G_m and H_m such that

$$G_m \equiv_m H_m$$

but G_m is strongly connected while H_m is not.

Ehrenfeucht Games

We define a **game** played between two players **Spoiler** and **Duplicator** on two database instances $\mathcal{I}$ and $\mathcal{J}$ over the same schema.

The game consists of m rounds, where in each round r:

1. **Spoiler** chooses an element from the active domain of one of the two instances: either $i_r \in adom(\mathcal{I})$ or $j_r \in adom(\mathcal{J})$.
2. **Duplicator** responds with an element from the active domain of the other instance: j_r or i_r.

At the end of play, we have two sequences $i_1, \ldots, i_m$ and $j_1, \ldots, j_m$.

Let σ denote the mapping $i_r \mapsto j_r$.

Ehrenfeucht Games *contd.*

If, for any relation R and any tuple u taken from $i_1, \ldots, i_m$ we have

$$u \in \mathcal{I}(R) \quad \text{if, and only if,} \quad \sigma(u) \in \mathcal{J}(R)$$

then **Duplicator** has won the game. Otherwise **Spoiler** has won.

Duplicator has a strategy for winning the m-round Ehrenfeucht game on $\mathcal{I}$ and $\mathcal{J}$ if, and only if,

$$\mathcal{I} \equiv_m \mathcal{J}.$$

Cycles

Take G_m to be a graph consisting of a single cycle of length 2^{m+1} and H_m to be a graph consisting of two disconnected cycles, each of length 2^m.

We can show $G_m \equiv_m H_m$

This proves that the transitive closure query is not definable in the relational calculus.

Database Theory: Lecture 6
Complex value model

Dr G.M. Bierman

www.cl.cam.ac.uk/Teaching/current/DBaseThy

History

- First Normal Form: Assumption in original Codd 1970 paper
- But...lots of application areas require more flexible model (even our Cinema database from lecture 1!)
- First proposal: Makinouchi [1977]
- Main activity in 80s:
  - Jaeschke & Schell [1982]
  - Thomas & Fischer [1983]
  - Abiteboul & Bidoit [1984]
  - Roth, Korth & Silberschatz [1986/8]
  - Schek & Scholl [1986]

Some differences, but basic idea: drop requirement that entries are atomic

Other names: NFNF, NF^2, ¬1NF, nested relational model, ...

Foundations

The notions of relation names, attributes and constants are as in the relational model

Sort	τ	$::=$	$\mathbf{dom}$	Base relation
		$\mid$	$\langle B_1 : \tau, \ldots, B_k : \tau_k \rangle$	Tuples $(k \geq 0)$
		$\mid$	$\{\tau\}$	Sets

- An element of a sort is called a **complex value**
- Useful shorthand: Often eliminate occurrences of $\mathbf{dom}$, e.g.

$$\langle A\colon \mathbf{dom}, B\colon \mathbf{dom}, C\colon \{\langle A\colon \mathbf{dom}, D\colon \mathbf{dom}\rangle\}\rangle = \langle A, B, C\colon \{\langle A, D\rangle\}\rangle$$

- Note that a complex value may belong to more than one sort

Extended language

- Relational algebra deals with sets of tuples. Thus complex value algebra should deal with sets of complex values.
- A complex value *relation* of sort τ is a finite set of values of sort τ.
- Common confusion: A complex value relation of sort $\langle A, B, C\rangle$ is a set of tuples over attributes ABC. The entire relation is also a complex value of sort $\{\langle ABC\rangle\}$.
- Note: Sort of a relation need not be a tuple.

ALGcv: Complex value algebra

ALGcv	q	$::=$	R	Base relation
		$\mid$	$\{a\}$	Constant values
		$\mid$	$q \cup q$	Union
		$\mid$	$q - q$	Set difference
		$\mid$	$\sigma_{B_i=a}(q)$	Select (constant)
		$\mid$	$\sigma_{B_i=B_j}(q)$	Select (attribute)
		$\mid$	$\sigma_{B_i\in B_j}(q)$	Select (attribute mem)
		$\mid$	$\sigma_{B_i=B_j.C}(q)$	Select (attribute comp)
		$\mid$	$\pi_{B_1,\ldots,B_k}(q)$	Projection
		$\mid$	$\mathsf{powerset}(q)$	Powerset
		$\mid$	$\mathsf{tupCreate}_{B_1,\ldots,B_k}(q_1,\ldots,q_k)$	Tuple
		$\mid$	$\mathsf{setCreate}(q)$	Set creation
		$\mid$	$\mathsf{tupDestroy}(q)$	Tuple destroy
		$\mid$	$\mathsf{setDestroy}(q)$	Set destroy

Semantics of ALGcv

$$
\begin{array}{rcl}
[\![R]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \mathcal{I}(R) \\
[\![\{a\}]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{a\} \\
[\![q_1 \cup q_2]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & [\![q_1]\!]_{\mathcal{I}} \cup [\![q_2]\!]_{\mathcal{I}} \\
[\![q_1 - q_2]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & [\![q_1]\!]_{\mathcal{I}} - [\![q_2]\!]_{\mathcal{I}} \\
[\![\sigma_{B_i=a}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{t \in [\![q]\!]_{\mathcal{I}} \mid t.B_i = a\} \\
[\![\sigma_{B_i=B_j}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{t \in [\![q]\!]_{\mathcal{I}} \mid t.B_i = t.B_j\} \\
[\![\sigma_{B_i\in B_j}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{t \in [\![q]\!]_{\mathcal{I}} \mid t.B_i \in t.B_j\} \\
[\![\sigma_{B_i=B_j.C}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{t \in [\![q]\!]_{\mathcal{I}} \mid t.B_i = t.B_j.C\} \\
[\![\pi_{B_1,\ldots,B_k}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{\langle B_1 : t.B_1, \ldots, B_k : t.B_k\rangle \mid t \in [\![q]\!]_{\mathcal{I}}\} \\
[\![\mathsf{powerset}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{v \mid v \subseteq [\![q]\!]_{\mathcal{I}}\} \\
[\![\mathsf{tupCreate}_{B_1,\ldots,B_k}(q_1,\ldots,q_k)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{\langle B_1 : v_1, \ldots B_k : v_k\rangle \mid v_1 \in [\![q_1]\!]_{\mathcal{I}}, \ldots, v_k \in [\![q_k]\!]_{\mathcal{I}}\} \\
[\![\mathsf{setCreate}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{[\![q]\!]_{\mathcal{I}}\} \\
[\![\mathsf{tupDestroy}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \{v \mid \langle A : v\rangle \in [\![q]\!]_{\mathcal{I}}\} \\
[\![\mathsf{setDestroy}(q)]\!]_{\mathcal{I}} & \stackrel{\text{def}}{=} & \bigcup [\![q]\!]_{\mathcal{I}}
\end{array}
$$

Sort judgements

$\mathcal{R} \vdash q : \tau$ means that q denotes a *relation* of sort τ, given database schema $\mathcal{R}$.

$$
\frac{R[\tau] \in \mathcal{R}}{\mathcal{R} \vdash R : \tau}
\qquad
\frac{}{\mathcal{R} \vdash \{a\} : \mathbf{dom}}
$$

$$
\frac{\mathcal{R} \vdash q_1 : \tau \quad \mathcal{R} \vdash q_2 : \tau}{\mathcal{R} \vdash q_1 \cup q_2 : \tau}
\qquad
\frac{\mathcal{R} \vdash q_1 : \tau \quad \mathcal{R} \vdash q_2 : \tau}{\mathcal{R} \vdash q_1 - q_2 : \tau}
$$

Sort judgements cont.

$$\frac{\mathcal{R} \vdash q\colon \tau \qquad \tau \equiv \langle \ldots, B_i\colon \mathbf{dom}, \ldots \rangle}{\mathcal{R} \vdash \sigma_{B_i=a}(q)\colon \tau}$$

$$\frac{\mathcal{R} \vdash q\colon \tau \qquad \tau \equiv \langle \ldots, B_i\colon \mathbf{dom}, \ldots, B_j\colon \mathbf{dom}, \ldots \rangle}{\mathcal{R} \vdash \sigma_{B_i=B_j}(q)\colon \tau}$$

$$\frac{\mathcal{R} \vdash q\colon \tau \qquad \tau \equiv \langle \ldots, B_i\colon \tau_i, \ldots, B_j\colon \{\tau_i\}, \ldots \rangle}{\mathcal{R} \vdash \sigma_{B_i \in B_j}(q)\colon \tau}$$

$$\frac{\mathcal{R} \vdash q\colon \tau \qquad \tau \equiv \langle \ldots, B_i\colon \tau_i, \ldots, B_j\colon \langle \ldots, C\colon \tau_i, \ldots \rangle, \ldots \rangle}{\mathcal{R} \vdash \sigma_{B_i=B_j.C}(q)\colon \tau}$$

Sort judgements cont.

$$\frac{\mathcal{R} \vdash q\colon \langle C_1\colon \tau_{C_1}, \ldots, C_k\colon \tau_{C_k} \rangle \qquad \{B_1, \ldots, B_n\} \subseteq \{C_1, \ldots, C_k\}}{\mathcal{R} \vdash \pi_{B_1,\ldots,B_n}(q)\colon \langle B_1\colon \tau_{B_1}, \ldots, B_n\colon \tau_{B_n} \rangle}$$

$$\frac{\mathcal{R} \vdash q_1\colon \tau_1 \ldots \mathcal{R} \vdash q_n\colon \tau_n}{\mathcal{R} \vdash \mathsf{tupCreate}_{B_1,\ldots,B_n}(q_1, \ldots, q_n)\colon \langle B_1\colon \tau_1, \ldots, B_n\colon \tau_n \rangle}$$

$$\frac{\mathcal{R} \vdash q\colon \langle B\colon \tau \rangle}{\mathcal{R} \vdash \mathsf{tupDestroy}(q)\colon \tau}$$

$$\frac{\mathcal{R} \vdash q\colon \tau}{\mathcal{R} \vdash \mathsf{setCreate}(q)\colon \{\tau\}} \qquad \frac{\mathcal{R} \vdash q\colon \{\tau\}}{\mathcal{R} \vdash \mathsf{setDestroy}(q)\colon \tau}$$

$$\frac{\mathcal{R} \vdash q\colon \tau}{\mathcal{R} \vdash \mathsf{powerset}(q)\colon \{\tau\}}$$

Generalized algebra

As in the relational algebra, there are a number of useful generalizations that can be coded up in ALGcv

- Complex Constants, e.g. $\{\langle A\colon a, B\colon \{b\} \rangle\}$
- Renaming
- Cross Product
- Join
- N-ary set creation

Aside: Clear that ALGcv subsumes relational algebra.

Further extensions: Nest and Unnest

(Simplified forms)

$$\frac{\mathcal{R} \vdash q\colon \langle A\colon \tau, B\colon \{\langle C\colon \tau', D\colon \tau'' \rangle\} \rangle}{\mathcal{R} \vdash \mathsf{unnest}_B(q)\colon \langle A\colon \tau, C\colon \tau', D\colon \tau'' \rangle}$$

$$\frac{\mathcal{R} \vdash q\colon \langle A\colon \tau, B\colon \tau', C\colon \tau'' \rangle}{\mathcal{R} \vdash \mathsf{nest}_{D=B,C}(q)\colon \langle A\colon \tau, D\colon \{\langle B\colon \tau', C\colon \tau'' \rangle\} \rangle}$$

Examples

Assume $\mathcal{R} = \{R[\langle A\colon \mathbf{dom}, A'\colon \mathbf{dom}\rangle], S[\langle B\colon \mathbf{dom}, B'\colon \{\mathbf{dom}\}\rangle]\}$.

1. Union of R and some constant tuples:

$$R \cup \{\langle A\colon 3, A'\colon 5\rangle, \langle A\colon 0, A'\colon 0\rangle\}$$

2. Tuples in S where the first component is contained in the second component

$$\sigma_{B \in B'}(S)$$

3. Cartesian product of R and S ?
4. Join of R and S on $A = B$

$$\sigma_{A=B}(R \times S)$$

Further examples

1. Renaming of R attributes to A_1, A_2

let $x =$ $\text{tupCreate}_{A_0,A_1,A_2}(R, \text{tupDestroy}(\pi_A(R)), \text{tupDestroy}(\pi_{A'}(R)))$
in $\pi_{A_1,A_2}(\sigma_{A_0.A=A_1}(\sigma_{A_0.A'=A_2}(x)))$

2. Unnesting of S on B' ?

CALCcv: Complex value calculus

Term	t	$::=$	a	Constant value
		$\mid$	x	Variable
		$\mid$	$x.A$	Tuple element
Literal	l	$::=$	$R(t)$	Relation atoms
		$\mid$	$t = t$	Equality
		$\mid$	$t \in t$	Membership
		$\mid$	$t \subseteq t$	Subset
Formula	F	$::=$	l	Literal
		$\mid$	$F \wedge F \mid F \vee F \mid \neg F$	Logical connectives
		$\mid$	$\exists x.F$	Powerset
		$\mid$	$\forall x.F$	Tuple
Query	q	$::=$	$\{x \mid F\}$	

Extensions to CALCcv

Similar to relational calculus

- **Complex constants** e.g. Assume $R\colon \langle A_1\colon \tau_1, \ldots, A_k\colon \tau_k\rangle$

$$R(u_1, \ldots, u_k) \stackrel{\text{def}}{=} \exists y.R(y) \wedge y.A_1 = u_1 \wedge \cdots \wedge y.A_k = u_k$$

- **Complex terms** e.g. $\{a, y\}$
- **Complex literals** e.g. $S \in T$
- **Queries as terms** e.g. $\{x \mid F\}$ where $fv(F) = \{x, y_1, \ldots, y_k\}$

Examples

Again assume
$\mathcal{R} = \{R[\langle A\colon \mathbf{dom}, A'\colon \mathbf{dom}\rangle], S[\langle B\colon \mathbf{dom}, B'\colon \{\mathbf{dom}\}\rangle]\}$.

1. Union of R and some constant tuples:

$$\{r \mid R(r) \lor r = \langle A\colon 3, A'\colon 5\rangle \lor r = \langle A\colon 0, A'\colon 0\rangle\}$$

2. Tuples in S where the first component is contained in the second component

$$\{s \mid S(s) \land s.B \in s.B'\}$$

3. Cartesian product of R and S

$$\{t \mid \exists r, s.R(r) \land S(s) \land t = \langle A\colon r.A, A'\colon r.A', B\colon s.B, B'\colon s.B'\rangle\}$$

4. Join of R and S on $A = B$

   ?

Transitive closure!

Recall $R\colon \langle A\colon \mathbf{dom}, A'\colon \mathbf{dom}\rangle$. Define

$$\begin{aligned} closed(x) &\stackrel{\text{def}}{=} \forall u, v, w.\langle A\colon u, A'\colon v\rangle \in x \land \langle A\colon v, A'\colon w\rangle \in x \\ &\quad \implies \langle A\colon u, A'\colon w\rangle \in x \end{aligned}$$

$$containsR(x) \stackrel{\text{def}}{=} \forall z.R(z) \implies z \in x$$

Then **transitive closure** can be defined as follows:

$$\{y \mid \forall x.closed(x) \land containsR(x) \implies y \in x\}$$

Recall that this can **not** be expressed in the relational calculus/algebra!!

Where did this extra expressive power come from?

Equivalence: From ALGcv to CALCcv

Given an ALGcv expression q we can define a CALCcv query $\{x \mid \|q\|_x\}$.

$$\begin{aligned} \|R\|_x &\stackrel{\text{def}}{=} R(x) \\ \|\{a\}\|_x &\stackrel{\text{def}}{=} x = a \\ \|\sigma_{B_i=a}(q)\|_x &\stackrel{\text{def}}{=} \|q\|_x \land (x.B_i = a) \\ \|\sigma_{B_i=B_j}(q)\|_x &\stackrel{\text{def}}{=} \|q\|_x \land (x.B_i = x.B_j) \\ &\ldots \\ \|\pi_{B_1,\ldots,B_k}(q)\|_x &\stackrel{\text{def}}{=} \exists y.(x = \langle B_1 : y.B_1, \ldots, B_k : y.B_k\rangle \land \|q\|_y) \\ \|q_1 \cap q_2\|_x &\stackrel{\text{def}}{=} \|q_1\|_x \land \|q_2\|_x \\ \|\mathsf{tupCreate}_{B_1,\ldots,B_k}(q_1, \ldots, q_k)\|_x &\stackrel{\text{def}}{=} \exists y_1 \ldots y_k.(x = \langle B_1 : y_1, \ldots, B_k : y_k\rangle \\ &\qquad \land \|q_1\|_{y_1} \land \cdots \land \|q_k\|_{y_k}) \\ \|\mathsf{tupDestroy}(q)\|_x &\stackrel{\text{def}}{=} \exists y.(\langle A\colon x\rangle = y \land \|q\|_y) \\ &\ldots \end{aligned}$$

Equivalence: From CALCcv to ALGcv

- This proof proceeds in exactly the same way as for the relational calculus to relational algebra
- (Again we will only consider domain-independent queries)
- The only extra complication is in the creation of the *active domains*

Constructing the active domains

Define a function $\tau^\star$ that maps a relation of sort τ to a relation of sort $\mathbf{dom}$ (essentially just returns the ground elements).

$$\begin{aligned}
\mathbf{dom}^\star &\stackrel{\text{def}}{=} \lambda x.x \\
\langle A_1 : \tau_1 \rangle^\star &\stackrel{\text{def}}{=} \lambda x.\tau_1^\star(\mathsf{tupDestroy}(x)) \\
\langle A_1 : \tau_1, \ldots A_k : \tau_k \rangle^\star &\stackrel{\text{def}}{=} \lambda x.((\langle A_1 : \tau_1 \rangle)^\star(\pi_{A_1}(x))) \cup \cdots \cup ((\langle A_k : \tau_k \rangle)^\star(\pi_{A_k}(x))) \\
\{\tau\}^\star &\stackrel{\text{def}}{=} \lambda x.\tau^\star(\mathsf{setDestroy}(x))
\end{aligned}$$

Then for a relation schema $\mathcal{R} = \{R_1[\tau_1], \ldots, R_k[\tau_k]\}$ and instance $\mathcal{I} = \{I_1, \ldots, I_k\}$

$$adom(q, \mathcal{I}) \stackrel{\text{def}}{=} \tau_1^\star(I_1) \cup \cdots \cup \tau_k^\star(I_k) \cup fe(q)$$

where $fe(q)$ are the free elements in q.

Variants of Complex Value Model

Nested relational model: Set and tuple constructors are required to *alternate*, e.g.

$\langle A, B, C : \{\langle D, E : \{\langle F, G \rangle\}\rangle\}\rangle$ ok

$\langle A, B, C : \langle D, E \rangle\rangle$ not ok

V-relation model: further restriction of the nested relational model where at least one component of every tuple sort must be *atomic*, e.g.

$\langle A, B, C : \{\langle D, E : \{\langle F, G \rangle\}\rangle\}\rangle$ ok

$\langle A, B, C : \{\langle E : \{\langle F, G \rangle\}\rangle\}\rangle$ not ok

Further assumption: atomic values in a tuple form a *key*

Work on the complex value model has naturally led to work on object-relational and object models (next lecture!).

Database Theory: Lecture 7
Object model

Dr G.M. Bierman

www.cl.cam.ac.uk/Teaching/current/DBaseThy/

Manifestos

By the late 80s following from work on Complex Value Models, and from work in the OO community, came suggestions for adding objects to database systems.

1. **The OODBMS manifesto** Atkinson, Bancilhon, DeWitt, Dittrich, Maier and Zdonik. 1989
2. **The third-generation database system manifesto** Committee for advanced DBMS function. 1990

(Date and Darwen have subsequently suggested a “third manifesto”)

The OODBMS manifesto

- Paper attempts to give a generic definition of an OODBMS
- It describes the main features and characteristics for a system to qualify as an OODBMS
- These were grouped as follows:
  1. **Mandatory** Features that your system *must* have
  2. **Optional** Features that might make your system *better*
  3. **Open** Features where you have free design space

These have had enormous influence both on OODBMSs and RDBMSs too!

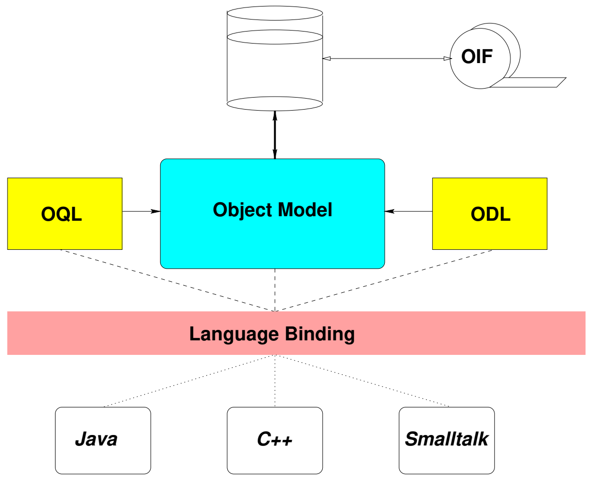
Mandatory commandments

- Thou shalt support **complex objects**
- Thou shalt support **object identity**
- Thou shalt **encapusulate** thine objects
- Thou shalt support **types** or **classes**
- Thine classes or types shalt **inherit** from their ancestors
- Thou shalt **not bind prematurely**
- Thou shalt be **computationally complete**
- Thou shalt be **extensible**
- Thou shalt **remember** thy data
- Thou shalt manage **very large databases**
- Thou shalt accept **concurrent users**
- Thou shalt **recover** from hardware and software failures
- Thou shalt have a simple way of **querying** data

ODMG: history

- Essentially an industrial response to the manifesto & a number of competing products
- Founded by Rick Cattell (Sun) in 1991 to produce an industry standard for object databases (but "avoid design by committee"!)
- Published various releases of the standard (1.1–Jan 94, 1.2–Jan 96, 2.0–May 97, 3.0–Jan 2000)
- Links with the OMG, and X3H2 (SQL) standard committee (amongst others)
- Now (2001-) refocused on the JDO design effort (Persistent objects for Java)
- Whilst now dormant: still an influential design and most OODBMS products are "compliant"

The ODMG architecture



ODL: Object Definition Language

- Language for specifying the structure of database objects
- Independent of your host programming language
- Class-based model
- (Superset of IDL - CORBA definition language)
- A class consists of
  - **Attributes** (cf. fields)
  - **Methods**
  - **Relationships** (specific to OODBMS)

ODL: Primitive types

A whole host of primitive (non-object) types, e.g.

- float, long, short, unsigned short, unsigned long, ...
- boolean, char, string
- date, time, timestamp, ...
- enumerations and structs (like in C)
- `set<T>`, `bag<T>`, `list<T>`, `array<T>`, `dictionary<T,T>`

(The latter are like the polymorphic constructor types in SML)

Example ODL definitions

```
 class Movie{
   attribute string title;
   attribute integer year;
   attribute enum Film {color,BW} filmType;
} ;

 class Star{
   attribute string name;
   attribute Struct Addr
          {string street, string city} address;
} ;
```


ODL methods

ODL allows declarations of methods (code is written in your favourite host language)

```
class Movie{
  attribute string title;
  attribute integer year;
  attribute enum Film {color,BW} filmType;
  float lengthInHours();
  set(string) starNames();
};
```

ODL relationships

ODL provides binary relationships.

```
class Movie{
  attribute string title;
  attribute integer year;
  attribute enum Film {color,BW} filmType;
  float lengthInHours();
  set(string) starNames();
  relationship Set<Star> stars
               inverse Star::starredin;
};

class Star{
  attribute string name;
  attribute Struct Addr
      {string street, string city} address;
  relationship Set<Movie> starredin
               inverse Movie::stars;
};
```


Classes

As in Java and C$^\sharp$, we allow single inheritance between classes, and provide *interfaces* with mutiple inheritance.

```
class Cartoon extends Movie{
   relationship Set<Star> voices
               inverse Star::VoiceIn;
};

class MurderMystery extends Movie{
  attribute string weapon;
};
```


Extents

ODL provides us with a handle on the collection of all current objects in a class: **extent**

```
class Movie (extent Movies){
  attribute string title;
  attribute integer year;
  attribute enum Film {color,BW} filmType;
  float lengthInHours();
  set(string) starNames();
  relationship Set<Star> stars
               inverse Star::starredin;
};
```

The extent name, e.g. `Movies`, can be used in queries, cf. relation name in SQL.

Expressivity

Consider the relational schema

$Guide$: `Cinema,Title,StartingTime` from Lecture 1. In ODL:

```
class GuideObject (extent Guide)
{
attribute string Cinema;
attribute string Title;
attribute Time StartingTime;
};
```

What about Complex Value Model?

More types

Rather confusingly, ODL also provides collection **classes**

- `Set<T>`, `Bag<T>`, `List<T>`, `Array<T>` and `Dictionary<T,T>`
- These are *classes*
- In fact they are **generic** classes (soon to be supported in Java/C$^\sharp$)

An introduction to OQL

- Based on SQL's `SELECT FROM WHERE` construct
- Looks like SQL-92
- Properly orthogonal:

  "OQL is a functional language whose operators can freely be composed, as long as the operands respect the type system."

Some example queries

1. `Movies;`
2. `select s.name from Stars as s;`
3. `select m.year from Movies as m where m.title="Magnolia";`
4. ```
   select s.name
   from   Movies as m,
          m.stars as s
   where  m.title="Magnolia";
   ```
5. ```
   select t.name
   from   (select m.stars
           from   Movies as m
           where  m.title="Magnolia") as t
   where  t.address.city="Hollywood";
   ```


Further features

1. **Casting**

   ```
   select  (Movie)m
   from    MurderMysteries as m
   where   m.weapon="AK47";
   ```

2. **Object creation**

   ```
   select  new Studio(owner: s.name)
   from    Stars as s;
   ```

Language binding

- Intention is to embed the database operations transparently into the host programming language
- Database programming now just looks like normal programming
- For most languages this is via an API
- Runtime language objects persist by backend magic

Java language binding

- ODMG specify a host of interfaces, e.g. for ODL types, OQL queries, e.g.

```
public interface OQLQuery{
    public void create(String query) ...
    public void bind(Object parameter) ...
    public Object execute ...
...
};
```

- Your vendor supplies the classes that implement them
- Java database programming now becomes just like normal programming

```
OQLQuery query;
query=getNewQuery();
query.create("select ... where p.salary>$1");
x=new Double(50000.0);
query.bind(x);
RichProfs=(DBag)query.execute();
```


Formalizing: MiniODMG

- One of the big problems with the ODMG was a lack of formalization
- So let's formalize! (Cutting edge [SIGMOD 2003!])
- We'll define
  - MiniODL
  - MiniOQL
- We'll assume that methods have been written in our favourite programming language, MiniDFlat.

MiniODL: Design choices

1. No interfaces
2. Limited primitive types
   - Just integers and booleans
3. No relationships
4. No collection classes (requires generics!)

MiniODL: Definitions

Class definition

$$
\begin{array}{lll}
cd & ::= & \texttt{class}\ \mathrm{C}_1\ \texttt{extends}\ \mathrm{C}_2 \\
 & & (\texttt{extent}\ e) \\
 & & \{ad_1 \ldots ad_k \\
 & & \ md_1 \ldots md_n\}
\end{array}
$$

Attribute definition

$$ad \quad ::= \quad \texttt{attribute}\ \phi\ a;$$

Method definition

$$md \quad ::= \quad \phi\ m\ (\phi_0\ x_0, \ldots, \phi_m\ x_m);$$

ODL type

$$\phi \quad ::= \quad \text{int} \mid \text{bool} \mid \text{C}$$

MiniODL: Example

```
class Employee extends Person
(extent Employees)
{ attribute int EmpID;
  attribute int GrossSalary;
  attribute Manager UniqueManager;
  int NetSalary (int TaxRate); }
```


MiniOQL: Definition

Query expression

q	$::=$	i	integer
	$\mid$	$\texttt{true} \mid \texttt{false}$	booleans
	$\mid$	x	identifier
	$\mid$	$\{q_0, \ldots, q_k\}$	set
	$\mid$	$q_1\ \texttt{sop}\ q_2$	set ops
	$\mid$	$q_1\ \texttt{iop}\ q_2$	int ops
	$\mid$	$q_1 {=} q_2$	int equality
	$\mid$	$q_1 {==} q_2$	object equality
	$\mid$	$\langle l_1 : q_1, \ldots, l_k : q_k \rangle$	record
	$\mid$	$q.l$	record access
	$\mid$	$\texttt{size}(q)$	set size

MiniOQL: Definition cont.

Query expression cont.

q	$::=$	$\ldots$	
	$\mid$	$(\mathrm{C})q$	cast
	$\mid$	$q.a$	attribute access
	$\mid$	$q.m(q_0, \ldots, q_k)$	method invocation
	$\mid$	$\texttt{new}\ \mathrm{C}(a_0 : q_0, \ldots, a_k : q_k)$	object creation
	$\mid$	$\texttt{if}\ q_1\ \texttt{then}\ q_2\ \texttt{else}\ q_3$	conditional
	$\mid$	$\texttt{select}\ q\ \texttt{from}\ x\ \texttt{in}\ q\ \texttt{where}\ q$	select

Type system for MiniOQL

MiniOQL type

$$\sigma \quad ::= \quad \phi \mid \mathsf{set}(\sigma) \mid \langle l_1 : \sigma_1, \ldots, l_k : \sigma_k \rangle$$

Type system

A query typing judgement is written

$$E; Q \vdash q : \sigma$$

where

- E is a map from extent identifiers to their type
- Q is a map from free identifiers of the query q to their type

Typing judgements

$$\frac{}{E; Q \vdash i : \mathsf{int}}\text{(Int)} \qquad \frac{}{E; Q \vdash \texttt{true} \mid \texttt{false} : \mathsf{bool}}\text{(Bool)}$$

$$\frac{}{E; Q, x : \sigma \vdash x : \sigma}\text{(Id)} \qquad \frac{}{E, e : \mathtt{C}; Q \vdash e : \mathsf{set}(\mathtt{C})}\text{(Extent)}$$

$$\frac{E; Q \vdash e : \mathsf{set}(\sigma)}{E; Q \vdash \texttt{size}(e) : \mathsf{int}}\text{(Size)}$$

$$\frac{E; Q \vdash q_0 : \sigma_0 \quad \cdots \quad E; Q \vdash q_k : \sigma_k \quad \forall i \in 0..k.\sigma_i \leq \sigma}{E; Q \vdash \{q_0, \ldots, q_k\} : \mathsf{set}(\sigma)}\text{(Set)}$$

$$\frac{E; Q \vdash q_1 : \mathsf{set}(\sigma) \quad E; Q \vdash q_2 : \mathsf{set}(\sigma)}{E; Q \vdash q_1 \cup q_2 : \mathsf{set}(\sigma)}\text{(Union)}$$

Typing judgements

$$\frac{E; Q \vdash q_1 : \sigma_1 \quad \cdots \quad E; Q \vdash q_k : \sigma_k}{E; Q \vdash \langle l_1 : q_1, \ldots, l_k : q_k \rangle : \langle l_1 : \sigma_1, \ldots, l_k : \sigma_k \rangle}\text{(Rec)}$$

$$\frac{E; Q \vdash q : \langle l_1 : \sigma_1, \ldots, l_k : \sigma_k \rangle \quad i \in 1..k}{E; Q \vdash q.l_i : \sigma_i}\text{(Rec access)}$$

$$\frac{E; Q \vdash q : \mathtt{C} \quad \mathtt{C} \leq \mathtt{C}'}{E; Q \vdash (\mathtt{C}')q : \mathtt{C}'}\text{(Upcast)}$$

$$\frac{E; Q \vdash q_2 : \mathsf{set}(\sigma) \quad E; Q, x : \sigma \vdash q_1 : \sigma_1 \quad E; Q, x : \sigma \vdash q_3 : \mathsf{bool}}{E; Q \vdash \texttt{select}\ q_1\ \texttt{from}\ x\ \texttt{in}\ q_2\ \texttt{where}\ q_3 : \mathsf{set}(\sigma_1)}\text{(Select)}$$

Observable nondeterminism

```
class P extends Object              class F extends Object
   ( extent Ps)                        ( extent Fs)
   { attribute int name;               { attribute int name;
};                                        attribute P   pal;};

SELECT (if size(Fs)<1
        then (new F(name:007,pal:p)).name
        else p.name)
FROM p in Ps;
```

Assume we have two P objects (001 and 002) and zero F objects.

What is the result of this query?

Even worse...

```
class P extends Object
   ( extent Ps)
   { attribute int name;
     F loop();          /* This method always loops! */
};


SELECT (if size(Fs)<1 and p.name=001
        then p.loop()
        else new F(name:007,pal:p)
FROM p in Ps;
```

What is the result of this query?

Can it be fixed?

Yes! In two ways:

1. Remove object creation from queries!
2. Make object creation invisible to current query (!)
3. Use a fancy type-based analysis (GMB)

Method support

Notice that MiniOQL allows you to invoke methods, which have been written in MiniDFlat.

- **ALL** DBMS manufacturers think you need this!
- What is the meaning of a query?
  - Dependent on meaning of arbitrary code!
  - How does this tie-in with semantic techniques so-far in DBT?
  - Queries can loop - which set is this denoted by?

Semantics of queries revisited

- Dependent on meaning of arbitrary code!

  Fine - we have operational semantics for Java/SML/...!

- How does this tie-in with semantic techniques in DBT?

  Maybe it's been wrong all along!

- Queries can loop - which set is this denoted by?

  Either move to domain theory; or use operational semantics!

Useful URLs

- `www.odmg.org`
- `www.service-architecture.com/articles`
- `java.sun.com/products/jdo`

Database Theory: Lecture 8
Semistructured model and XML

Dr G.M. Bierman

`www.cl.cam.ac.uk/Teaching/current/DBaseThy`

Schema-less data

- Underlying assumption so far: **Data has a fixed schema declared in advance**
- E.g. `CREATE TABLE` in SQL, ODL in ODMG object databases
- Obvious question: *What happens if we drop this assumption?*
- Note: This predates the web-based applications!
- First prototype system: TSIMMIS/Lore at Stanford [1995]
- [Aside: Another assumption is *persistent* data sets. Dropping this yields a data stream model—a HOT area!]

Is this a good idea?

The world is really quite unstructured, for example:

- Data exchange formats (pre-XML)
- The World-Wide Web
- ACeDB - a database used by biologists (from Sanger Centre, Cambridge)

Is this a good idea?

To query the database one currently needs to understand the schema

But...lots of users either don't understand the schema or don't want to know about the schema

- Where in the database is there information on “Magnolia”?
- Are there integers less than -256?
- Which database objects have attribute names beginning with “Name”?

So let's move to schema-less data model!

Semistructured data

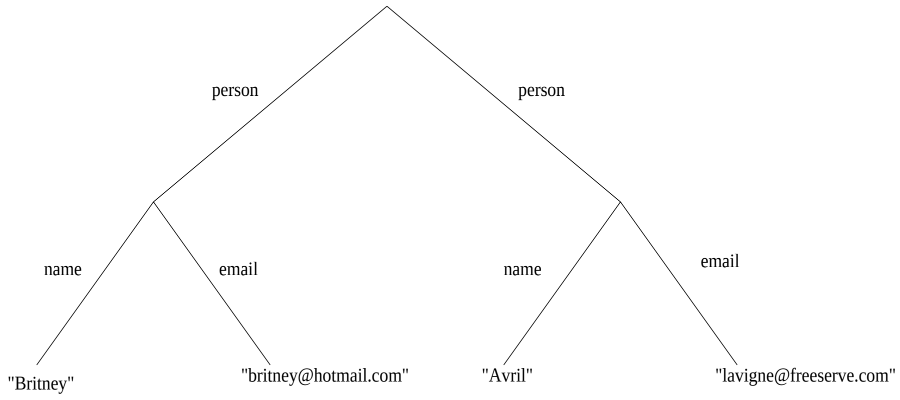
- Rather than devising a syntax for *types*, we need a general syntax for *values*
- First start with simple idea: **records**

  `{name:"Britney", email:"bspears@hotmail.com"}`
- Thus a value can be a collection of label-value pairs, e.g.

  `{name:{first:"Britney",second:"Spears"}, email:"bspears@hotmail.com"}`
- (We'll drop the standard convention that labels must be unique)

```
{person:{name:"Britney", email:"bspears@hotmail.com"},
 person:{name:"Avril", email:"lavigne@freeserve.com"}}
```


SSD and graphs



Notice that these values can be represented using **edge-labelled trees**

Relational model

It's pretty obvious that relational database instances can be represented using this model.

For example $\mathcal{R} = \{R[A, B, C], S[D, E]\}$. An instance could be stored as

```
{R:{row:{A:"Britney",B:"St Johns", C:"E1(a)"},
    row:{A:"Christina",B:"Trinity",C:"F7"}},
 S:{row:{D:"Myleene",E:"Thursdays"},
    row:{D:"Avril",E:"Fridays"}
    row:{D:"Britney",E:"Saturdays"}}
}
```

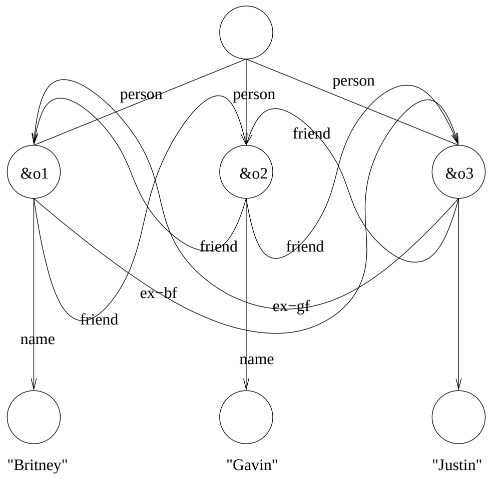

Object model

- Objects have *identity*
- Let's extend our semi-structured model to label values with **oids** and allow oids to be values, e.g.

```
{person: &o1{name:"Britney",
             friend: &o2,
             ex-bf: &o3},
 person: &o2{name:"Gavin",
             friend: &o2,
             friend: &o3},
 person: &o3{name:"Justin",
             friend: &o2,
             ex-gf: &o1}
}
```


Graphs!

Drawing this example graphically: Data is now a **graph**!



Semi-structured data model

SSD	s	$::=$	v	Value
		$\mid$	&o v	Value with identity
		$\mid$	&o	Object identity
Value	v	$::=$	a	Atomic values
		$\mid$	c	Complex value
Complex value	c	$::=$	$\{l : s, \ldots, l : s\}$	

Coding up ODMG schema

```
class Movie (extent Movies){
  attribute string title;
  attribute integer year;
  attribute enum Film {color,BW} filmType;
  relationship Set<Star> stars
               inverse Star::starredin;
};

class Star (extent Stars){
  attribute string name;
  attribute Struct Addr
      {string street, string city} address;
 relationship Set<Movie> starredin
               inverse Movie::stars;
};
```


Some example SSD compliant data

```
{Movies:{Movie:&m01{title:"Magnolia",
                    year:1999,
                    film:{colour},
                    stars:{Star:&s01,Star:&s02}},
         Movie:&m02{title:"Safe",
                    ...}},
 Stars:{Star:&s01{name:"Julianne Moore",
                  Addr:{street:"Sunset Blvd",city:"LA"},
                  starredin:{Movie:&m01,Movie:&m02}},
        Star:&s02{name:"Tom Cruise",
                  ...}}
}
```

Thus it is easy to embed ODMG data into SSD model

Expressivity cont.

- We have seen that we can code up relational and object model instances into SSD. What about the other direction?
- **Not so easy!** The point of the SSD model is its **flexibility**, e.g.

```
{Friend:{Name:"Britney",
         Tel:337890},
 Friend:{Name:"Emma",
         Email:"baby@spicegirls.com"},
 Friend:{Email:"sc@popstars.com",
         Tel:898989},
 Friend:{Name:"Myleene"}
}
```

- **SSD model offers a very flexible data model!**
- *How might we store SSD values in a relational model?*

SSD systems

- A number of these flourished in the mid to late 1990s.
- Most notable work at Stanford:
  - TSIMMIS [1995-]
  - Lore [1997-]
- Other interesting systems include ACeDB:
  - Built at the Sanger Centre (Cambridge)
  - Originally to store genetic data about C. elegans organism
  - Essentially a SSD model (has a schema language, but allows labels to be missing)

XML

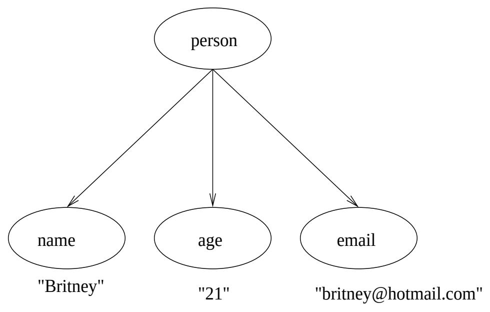
- Standard for data exchange on web
- Textual format, consisting of elements wrapped with matching tags (mark-up), e.g.

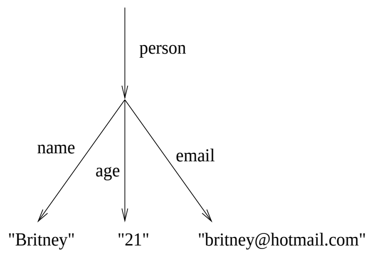
```
<person><name>Britney</name>
        <age>21</age>
        <email>britney@hotmail.com</email>
</person>
```

- This clearly has a corresponding SSD value

```
{person:{name:"Britney",age:21,email:"britney@hotmail.com"}}
```


XML graph model





- XML is a **node**-labelled graph
- SSD is an **edge**-labelled graph

Building graphs in XML

XML provides special *attributes* to build graphs: `ID` and `IDREF`, e.g.

```
<IMDB>
 <Movies>
  <Movie id="&m01">
   <title>Magnolia</title>
   <year>1999</year>
   <film>colour</film>
   <stars><star idref="&s01"/>
          <star idref="&s02"/>
   </stars>
  </Movie>
  <Movie id="&m02">
   <title>Safe</title>
    ...
  </Movie>
...
 </Movies>
 <Stars>
```



```
  <Star id="&s01">
   <name>Julianne Moore</name>
   <Addr><street>Sunset Blvd</street><city>LA</city></Addr>
   <starredin><Movie idref="&m01"/><Movie idref="&m02"/>
  </Star>
  <Star id="&s02">
   <name>Tom Cruise</name>
   ...
  </Star>
 </Stars>
</IMDB>
```

Querying XML

- There were a series of proposals, XML-QL, XQL, UnQL, Quilt...
- Lead to current proposal: **XQuery**
- XQuery consists of two parts
  1. **XPath**: A language for describing nodes in an XML tree
  2. **XQuery**: Query-like language surrounding XPath

XPath

- `http://www.w3.org/TR/xpath`
- Building block for other W3C standards (not just XQuery) e.g. XLink, XPointer etc.

XPath

- Inspiration is that navigation of a tree is like navigating a Unix-style directory
- All paths start from a **context node**
- Result is (approximately) the set of nodes reachable from the context node using the given path expression

Example XML document

```
<bib>
   <book>  <publisher> Addison-Wesley </publisher>
           <author> Serge Abiteboul </author>
           <author> <first-name> Rick </first-name>
                    <last-name> Hull </last-name>
           </author>
           <author> Victor Vianu </author>
           <title> Foundations of Databases </title>
           <year> 1995 </year>
   </book>
   <book price="55">
         <publisher> Freeman </publisher>
         <author> Jeffrey D. Ullman </author>
         <title> Principles of Database and Knowledge Base Systems</title>
         <year> 1998 </year>
   </book>
</bib>
```

Example XPath

Query: `/bib/book/year`

Result: `<year>1995</year> <year>1998</year>`

Query: `/bib/paper/year`

Result:

Query: `//author`

Result: `<author> Serge Abiteboul </author>`

`<author> <first-name> Rick </first-name>...`

Query: `/bib//first-name`

Result: `<first-name> Rick </first-name>`

XPath: Functions

Query: `/bib/book/author/text()`

Result: `Serge Abiteboul`

`Jeffrey D. Ullman`

Functions available:

- `text()` – matches the text value
- `node()` – matches any node
- `name()` – returns the name of the tag

XPath: Wildcard

Query: `//author/*`

Result: ?

XPath: Attribute Nodes

Query: `/bib/book/@price`

Result: 55

NB: `price` is an attribute

XPath: Qualifiers

Query: `/bib/book/author[first-name]`

Result: `<author><first-name> Rick </first-name>`
`<last-name> Hull </last-name></author>`

Query: `/bib/book[@price<"60"]`

Query: `/bib/book[author/@age<"25"]`

Thus we get a primitive form of querying (although result is a set of nodes, not valid XML)

XPath summary

`bib`	matches a `bib` element
`*`	matches any element
`/`	matches the root element
`/bib`	matches a `bib` element under root
`bib/paper`	matches a `paper` in `bib`
`bib//paper`	matches a `paper` in `bib`, at any depth
`//paper`	matches a `paper` at any depth
`paper|book`	matches a `paper` **or** a `book`
`@price`	matches a `price` attribute
`bib/book/@price`	matches `price` attribute in `book`, in `bib`

XQuery

XPath is central to XQuery. In addition, XQuery provides

- XML glue to turn XPath results back into XML
- Variables to communicate between XPath and XQuery
- Fancy database features, e.g. joins, aggregates etc.
- Fundamental structure:

  `for let where return`

Simple XQuery

```
FOR    $x IN document("bib.xml")/bib/book
WHERE  $x/year > 1995
RETURN $x/title
```

Returns:

```
<title> abc </title> <title> def </title>
<title> ghi </title>
```

Simple XQuery

```
FOR $a IN distinct(document("bib.xml")
                   /bib/book[publisher="Morgan Kaufmann"]/author)
RETURN <result>
        $a,
        FOR $t IN /bib/book[author=$a]/title
        RETURN $t
       </result>
```

`distinct` is a function that removes duplicates

Bindings

- `FOR $x IN e` – binds `$x` to each element in the expression `e`
- `LET $x = e'` – binds `$x` to the entire expression `e'`

Example LET/WHERE query

```
<big_publishers>
       FOR    $p IN distinct(document("bib.xml")//publisher)
       LET    $b := document("bib.xml")/book[publisher = $p]
       WHERE  count($b) > 100
       RETURN $p
</big_publishers>
```

`count` is an aggregate function that returns the number of elements

XQuery

- Lots of other features (sorting, filters, recursive function definitions, conditionals, ...)
- **Really cool thing:**
  - **http://www.w3.org/TR/query-semantics/**
  - This contains an operational semantics and type system for the core fragment of XQuery!
  - Industrial application of 1B Semantics material!