From complex values to objects

A database perspective

Guillaume Raschia — Université de Nantes

Last update: November 17, 2012

[Source : E. Schallehn, Univ. of Magdeburg]

[Source : T. Calders, Univ. of Eindhoven]

[Source : M. Gertz, UC Davis (now Univ. of Heidelberg)]

[Source : B. Signer, Vrije Universiteit Brussel]

A very first example

- Class book
  - title
  - set of authors
  - publisher
  - set of keywords
- Easy to model in any programming language
- Tricky in relational database!

Basic proposal

- Either we ignore the normalization...

Title	Author	Publisher	Keyword
FoD	S. Abiteboul	Addison-Wesley	Database
FoD	R. Hull	Addison-Wesley	Database
FoD	V. Vianu	Addison-Wesley	Database
FoD	S. Abiteboul	Addison-Wesley	Logic
FoD	R. Hull	Addison-Wesley	Logic
FoD	V. Vianu	Addison-Wesley	Logic
TCB	J.D. Ullman	Pearson	Database
⋮	⋮	⋮	⋮

- Key: (Title, Author, Keyword)
- Not in 2NF since Title $\longrightarrow$ Publisher

Intermediate state

- ... Or we go to 3NF, BCNF

Title	Publisher
FoD	Addison-Wesley

Title	Author	Keyword
FoD	S. Abiteboul	Database
FoD	R. Hull	Database
FoD	V. Vianu	Database
FoD	S. Abiteboul	Logic
FoD	R. Hull	Logic
FoD	V. Vianu	Logic

- But we still ignore the multivalued dependencies...
- Title $\twoheadrightarrow$ Author and Title $\twoheadrightarrow$ Keyword

About MVD's and 4NF

- MVD: full constraint on relation

Definition (Multi-Valued Dependency)

Let R be a relation of schema $\{X, Y, Z\}$; $X \twoheadrightarrow Y$ holds whenever (x, y, z) and (x, t, u) exist in R implies that (x, y, u) and (x, t, z) should also exist in R

Example

- Analysis: Department {Building} {Employee {Telephone}}
- MVD's = {Department $\twoheadrightarrow$ Building, Employee $\twoheadrightarrow$ Telephone}

About MVD's and 4NF (cont'd)

MVD Properties in $R(X, Y, Z)$

- $X \twoheadrightarrow Y \Rightarrow X \twoheadrightarrow Z$
- $X \rightarrow Y \Rightarrow X \twoheadrightarrow Y$
- $X \twoheadrightarrow R - X$ always holds (trivial MVD)

Definition (4NF)

For every non trivial MVD $X \twoheadrightarrow Y$ in R, then X is a superkey

Losseless-join decomposition

A decomposition of R into (X, Y) and (X, Z) is a **losseless-join decomposition** iff $X \twoheadrightarrow Y$ holds in R

The ultimate schema

- . . . Or we go to 4NF

Title	Publisher
FoD	Addison-Wesley

Title	Author
FoD	S. Abiteboul
FoD	R. Hull
FoD	V. Vianu

Title	Keyword
FoD	Database
FoD	Logic

Pros & Cons

- 4NF design
  - requires many joins in queries (performance pitfall)
  - and loses the big picture of entities
- 1NF relational view
  - eliminates the need for users to perform joins
  - but loses the one-to-one correspondence between tuples and objects
  - has a large amount of redundancy
  - and could yield to insertion, deletion, update anomalies

Contents

Preamble

Alice:	Complex values?
Riccardo:	We could have used a different title: nested relations, complex objects, structured objects...
Vittorio:	...N1NF, NFNF, NF2, NF^2, V-relation... I have seen all these names and others as well.
Sergio:	In a nutshell, relations are nested within relations; something like Matriochka relations.
Alice:	Oh, yes. I love Matriochkas.

FoD: chap. 20, p. 508

Beyond the Relational Model

- Theoretical extensions of the RM
  - Nested relations: NF^2
  - Type constructors and free combination: eNF^2
- New Requirements
  - Operations as extension to relational algebra
  - Normal form to provide consistency
- Today, part of SQL3 and commercial systems

The NF^2 Database Model

NF^2 = NFNF = Non First Normal Form

Principle

NF^2 relations permit **complex values** whenever we encounter atomic, i.e. indivisible, values

- Violates first normal form
- Allows more intuitive—let say *conceptual*—modeling for applications with complex data
- Preserves mathematical foundations of relational model

From Flatland to Lineland[1]...

- Relation schema in 1NF: τ

$$\tau := \langle A_1 : \mathrm{dom}, \ldots, A_k : \mathrm{dom} \rangle$$

  - Sort constructors: **tuple** and—finite—**set**
  - Construction pattern: set(tuple(dom*))

- $\neg$1NF: much more combinations

$$\tau := \mathrm{dom} \mid \langle A_1 : \tau, \ldots, A_k : \tau \rangle \mid \{\tau\}$$

- We denote by $[\![\tau]\!]$ the set of complex values of sort τ

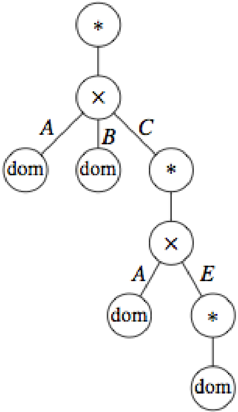
[1] **Flatland, a Romance of Many Dimensions**. Edwin A. Abbott (1884).

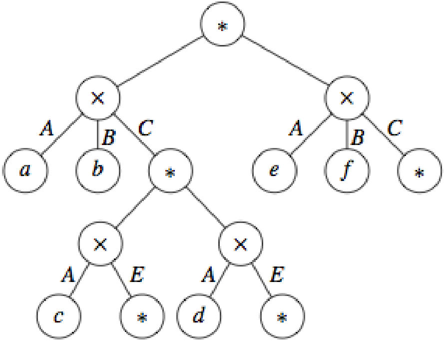
Examples

Sort	Complex value
dom	a
$\{\mathrm{dom}\}$	$\{a, b, c\}$
$\{\{\mathrm{dom}\}\}$	$\{\{a, b\}, \{a\}, \{\}\}$
$\langle A : \mathrm{dom}, B : \mathrm{dom}\rangle$	$\langle A : a, B : b\rangle$
$\langle A : \langle B : \mathrm{dom}\rangle\rangle$	$\langle A : \langle B : b\rangle\rangle$
$\{\langle A : \mathrm{dom}, B : \mathrm{dom}\rangle\}$	$\{\langle A : a, B : b\rangle, \langle A : a, B : b\rangle\}$
$\langle A : \{\langle B : \mathrm{dom}\rangle\}\rangle$	$\langle A : \{\langle B : b\rangle, \langle B : c\rangle\}\rangle$

With trees. . .

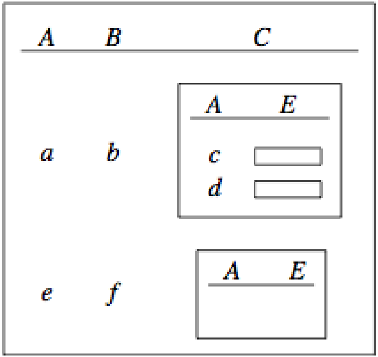
Sort and complex value as **finite trees**





With tables as usual...

Complex value as **table**



Side note: also of sort $\langle A, B, C : \{\langle A, E : \{\emptyset\}\rangle\}\rangle$

Database instance

Definition (NF^2 Schema)

A schema $R\!:\!\tau$ is a relation name R with $\mathrm{sort}(R) = \tau$

- Extend to database schema $\mathcal{R}$ as a set of relation schemes

Definition (NF^2 Relation)

A relation $\mathtt{I}(R)$ of schema $R\!:\!\tau$ is a finite set of values of sort τ

Definition (NF^2 Database instance)

A database instance $\mathtt{I}$ over schema $\mathcal{R}$ is a mapping

$$\begin{array}{rlcl} \mathtt{I}: & \mathcal{R} & \longrightarrow & \mathcal{P}^{[\![\mathrm{sort}(\mathcal{R})]\!]} \\ & R\!:\!\tau & \longmapsto & \mathtt{I}(R) \end{array}$$

Running example

Abuse of notation

- R stands either for schema or instance of a relation, wherever it is unambiguous
- X is a shorthand for $X\!:\!\text{dom}$ wherever it is applicable

Database

Instance J of schema $\mathcal{R} = \{R_1, R_2, R_3\}$ with

$$\text{sort}(R_1) = \text{sort}(R_3) = \langle A, B\!:\!\{\langle A_1, A_2\rangle\}\rangle$$
$$\text{sort}(R_2) = \langle A, A_1, A_2\rangle$$

Running example (cont'd)

<table>
<tr><th>A</th><th>B</th></tr>
<tr><td>d_1</td><td><table><tr><th>A_1</th><th>A_2</th></tr><tr><td>d_1</td><td>d_2</td></tr><tr><td>d_3</td><td>d_4</td></tr></table></td></tr>
<tr><td>d_1</td><td><table><tr><th>A_1</th><th>A_2</th></tr><tr><td>d_3</td><td>d_4</td></tr><tr><td>d_5</td><td>d_6</td></tr></table></td></tr>
<tr><td>d_2</td><td><table><tr><th>A_1</th><th>A_2</th></tr><tr><td>d_1</td><td>d_3</td></tr><tr><td>d_2</td><td>d_4</td></tr></table></td></tr>
</table>

$\mathbf{J}(R_1)$

A	A_1	A_2
d_1	d_1	d_2
d_1	d_3	d_4
d_1	d_5	d_6
d_2	d_1	d_3
d_2	d_2	d_4

$\mathbf{J}(R_2)$

<table>
<tr><th>A</th><th>B</th></tr>
<tr><td>d_1</td><td><table><tr><th>A_1</th><th>A_2</th></tr><tr><td>d_1</td><td>d_2</td></tr><tr><td>d_3</td><td>d_4</td></tr><tr><td>d_5</td><td>d_6</td></tr></table></td></tr>
<tr><td>d_2</td><td><table><tr><th>A_1</th><th>A_2</th></tr><tr><td>d_1</td><td>d_3</td></tr><tr><td>d_2</td><td>d_4</td></tr></table></td></tr>
</table>

$\mathbf{J}(R_3)$

Exercises 1/2

1. Definitions

MVD, 4NF, NF^2, Complex value, NF^2 Schema, NF^2 Relation, NF^2 Db instance

2. True or False?

i) FD's are MVD's.
ii) 4NF generalizes BCNF.
iii) Relation $R \subseteq \{\vartheta \mid \vartheta \vdash \tau\}$ is of sort $\{\tau\}$.
iv) Set vertex in a sort tree may have many children.
v) Set vertex in a CV tree may have many children.
vi) Type inference from CV yields one single sort.

Exercises 2/2

3. Sorts and complex values

Draw the tree of each sort and complex value from the *Examples* slide.

4. Problem ✐

Consider a (flat) relation R of sort $\langle$email, location, post, friend_email, friend_location$\rangle$ and the MVD email location $\twoheadrightarrow$ post. Prove that the same information can be stored in a complex value relation of sort

$$\langle \text{email}, \text{location}, \text{posts} : \{dom\}, \text{friends} : \{\langle \text{friend\_email}, \text{friend\_location}\rangle\}\rangle$$

Discuss pros of this alternative representation (w.r.t. size and update anomalies).

Overview

Chapter Nested Tables

1. From NF^2 to Nested Tables
2. SQL3 Transcription of NF^2
3. Design and Normalization of Nested Tables

A popular restriction

Definition (Nested relation)

A nested relation is a NF^2 relation where **set** and **tuple** constructors are required to alternate

Examples

$$
\begin{array}{lll}
\tau_1 = & \langle A, B, C : \{\langle D, E : \{\langle F, G\rangle\}\rangle\}\rangle & \text{Ok} \\
\tau_2 = & \langle A, B, C : \{\langle E : \{\langle F, G\rangle\}\rangle\}\rangle & \text{Ok} \\
\tau_3 = & \langle A, B, C : \langle D, E : \{\langle F, G\rangle\}\rangle\rangle & \text{No!} \\
\tau_4 = & \langle A, B, C : \{\{\langle F, G\rangle\}\}\rangle & \text{No!}
\end{array}
$$

One more *real-life* example to take away

Relation Dpts of sort
$\langle \text{Dpt}, \text{Emps}\colon \{\langle \text{SSN}, \text{Name}, \text{Tels}\colon \{\langle \text{Tel} \rangle\}, \text{Salary}\rangle\}\rangle$

Department	Employees			
	SSN	Name	Telephones	Salary
			Tel	
Computer Science	4711	Todd	038203-12230	6,000
			0381-498-3401	
	5588	Whitman	0391-334677	6,000
			0391-5592-3452	
	7754	Miller		550
	8832	Kowalski		2,800
Mathematics	6834	Wheat	0345-56923	750

About nested relations

Nested relations vs. N1NF-relations

Cosmetic restriction only

Size of nested relations

$\mathcal{O}(2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}})$ with n being the size of the active domain of R and *the tower of 2* equals depth of R (nested levels)

Reminder: Size of flat relations is polynomial

Languages for nested relations

Logic

Mainly extend RC to **variables denoting sets**

$$\{t.\mathrm{Dpt} \mid \mathrm{Dpts}(t) \ \wedge \ \forall X, u : (t.\mathrm{Emps} = X \ \wedge \\ u \in X \rightarrow u.\mathrm{Salary} \leq 5,000)\}$$

With queries as terms:

$$\{t.\mathrm{Dpt} \mid \mathrm{Dpts}(t) \ \wedge \ t.\mathrm{Emps} \subseteq \{u \mid u.\mathrm{Salary} \leq 5,000\}\}$$

Operations on nested relations

The usual way

$$\sigma_{A=d_1}(R_1) \quad \text{and} \quad \pi_A(R_1)$$

$$R_1 \bowtie R_2$$

$$R_1 - R_3 \quad \text{and} \quad R_1 \cup R_3$$

Straightforward extensions

$$\sigma_{B=C}(R_1 \bowtie \rho_{B\to C}(R_3))$$

$$\sigma_{B\subset C}(R_1 \bowtie \rho_{B\to C}(R_3))$$

$$\pi_{B.A_1}(R_1)$$

$$\sigma_{A\in B.A_1}(R_1)$$

Nested relational algebra

- $\cup$ $-$ π $\bowtie$ nearly as in relational algebra
- σ: condition extended to support
  - **Relations as operands** (instead of constants in dom)
  - **Set operations** like $\theta \in \{\in, \subseteq, \subset, \supset, \supseteq\}$
- **Recursively structured** operation parameters, e.g.
  - π: nested projection attribute lists
  - σ: selection conditions on nested relations

Nested relational algebra (cont'd)

- Additional operations: Nest (ν) and Unnest (μ)
  - $\mu_{\$k}(\mathtt{I}(R))$: remove nesting from k^{th} column of $\mathtt{I}(R)$
  - $\nu_{\$1,\ldots,\$k}(\mathtt{I}(R))$: nest columns $1,\ldots k$ of $\mathtt{I}(R)$
  - There exists an equivalent named flavor
- A curiosity: The **Powerset** operator

$$\Omega(\mathtt{I}(R)) = \{\vartheta \mid \vartheta \subseteq \mathtt{I}(R)\}$$

  - Ω extends algebra up to **reachability** (eq. Datalog)

Unnest operator

- Suppose schema $\mathcal{R} = \{R, S\}$ with sorts:

$$\mathrm{sort}(R) = \langle A_1 : \tau_1, \ldots, A_k : \tau_k, B : \{\langle A_{k+1} : \tau_{k+1}, \ldots, A_n : \tau_n \rangle\}\rangle$$
$$\mathrm{sort}(S) = \langle A_1 : \tau_1, \ldots, A_k : \tau_k, A_{k+1} : \tau_{k+1}, \ldots, A_n : \tau_n \rangle$$

- The unnest operator is defined as follows:

$$\begin{aligned}\mu_B(\mathtt{I}(R)) = \{ & \langle A_1 : x_1, \ldots, A_n : x_n \rangle \mid \\ & \exists y : \langle A_1 : x_1, \ldots, A_k : x_k, B : y \rangle \in \mathtt{I}(R) \land \\ & \langle A_{k+1} : x_{k+1}, \ldots, A_n : x_n \rangle \in y \}\end{aligned}$$

- Example: $\mu_B(R_1) = R_2$

Nest operator

$$
\begin{aligned}
\nu_{B=(A_{k+1},\ldots,A_n)}(\mathtt{I}(S)) = \{\langle A_1 : x_1, \ldots, A_k : x_k, B : y\rangle \mid \\
y = \{\langle A_{k+1} : x_{k+1}, \ldots, A_n : x_n\rangle \mid \\
\langle A_1 : x_1, \ldots, A_n : x_n\rangle \in \mathtt{I}(S)\} \wedge \\
y \neq \emptyset\}
\end{aligned}
$$

- Example: $\nu_{B=(A_1,A_2)}(R_2) = R_3$

Nest & Unnest

A	B	C
1	2	7
1	3	6
1	4	5
2	1	1

$$\xrightarrow{\nu_{D=(B,C)}(S)}$$

$$\xleftarrow{\mu_D(R)}$$

<table>
<tr><th rowspan="2">A</th><th colspan="2">D</th></tr>
<tr><th>B</th><th>C</th></tr>
<tr><td rowspan="3">1</td><td>2</td><td>7</td></tr>
<tr><td>3</td><td>6</td></tr>
<tr><td>4</td><td>5</td></tr>
<tr><td>2</td><td>1</td><td>1</td></tr>
</table>

About duality of Nest & Unnest

- The following statement holds:

$$\mu_B(\nu_{B=(A_1A_2)}(R_2)) = R_2$$

- However

$$\nu_{B=(A_1A_2)}(\mu_B(R_1)) \neq R_1$$

- Unnest is the **right inverse** of nest: $\mu_A \circ \nu_{A=\alpha} \equiv \mathrm{Id}$
- Unnest is **not information preserving** (one-to-one) and so has no right inverse

About duality of Nest & Unnest (cont'd)

Unnesting not generally reversible

A	D	
	B	C
1	2	7
	3	6
1	4	5
2	1	1

$\xrightarrow[\mu_D(R)]{}$

A	B	C
1	2	7
1	3	6
1	4	5
2	1	1

$\xrightarrow[\nu_{D=(B,C)}(S)]{}$

A	D	
	B	C
1	2	7
	3	6
	4	5
2	1	1

Nesting in queries

Flat-Flat Theorem

Let Q be a nested relational algebra expression;

- Q takes a non-nested relation as input
- Q produces a non-nested relation as output

Then, Q can be rewritten as a **regular relational algebra expression** (i.e., w/o nesting)

Nesting in queries (cont'd)

Result is actually stronger for query Q

Nested Query Theorem

Assume a d_1-nested relation as input and a d_2-nested relation as output; there is no need for intermediate results having depth greater than $\max(d_1, d_2)$

What for?

- Can be used by query optimizers
- No need to introduce intermediate nesting
- Standard techniques for query evaluation

NF^2 concepts in SQL3

- SQL:1999 introduced **tuple type constructor ROW**
- Only few changes to type system in SQL:2003
  - **Bag type constructor MULTISET**
  - XML data types
- Implementations in commercial DBMS most often do NOT comply with standard!

ROW type constructor

- `ROW` implements **tuple** type constructor

Example

```
CREATE ROW TYPE AddressType (
                             Street   VARCHAR(30),
                             City     VARCHAR(30),
                             Zip      VARCHAR(10) );
CREATE ROW TYPE CustomerType (
                             Name     VARCHAR(40),
                             Address  AdressType );

CREATE TABLE Customer OF TYPE CustomerType
                             ( PRIMARY KEY Name );
```

ROW type constructor (cont'd)

- Insertion of records requires call to **row constructor**

```
INSERT INTO Customer
VALUES( 'Doe', ROW( '50 Otages', 'Nantes', '44000' ) );
```

- Component access by usual dot '.' notation with field parenthesis ($\neq$ table prefix)

```
SELECT C.Name, (C.Address).City FROM Customer C;
```

MULTISET type constructor

- SQL:2003 MULTISET implements set/bag type constructor
- Can be combined with ROW type constructors
- Allows creation of nested tables (NF^2)

```
CREATE TABLE Department (
    Name VARCHAR(40),
    Buildings INTEGER MULTISET,
    Employees ROW( Firstname VARCHAR(30),
                   Lastname  VARCHAR(30),
                   Office    INTEGER ) MULTISET );
```

MULTISET type constructor (cont'd)

Operations

- `MULTISET` constructor
- `UNNEST` implements μ
- `COLLECT`: special aggregate function to implement ν
- `FUSION`: special aggregate function to build union of aggregated multisets
- `MULTISET UNION|INTERSECT|EXCEPT`
- `CARDINALITY` for size
- `SET` eliminates duplicates
- `ELEMENT` converts singleton to a tuple (row) expression

MULTISET type constructor (cont'd)

Predicates

- `MEMBER`: $x \in E$
- `SUBMULTISET` multiset containment: $S \subseteq E$
- `IS [NOT] A SET` test whether there are duplicates or not

```
SELECT D.Name FROM Department D
WHERE CARDINALITY(D.Buildings) >= 2 AND
      D.Employees IS A SET;
```

MULTISET type constructor (cont'd)

Insert and **Update** statements

```
INSERT INTO Department
VALUES( 'Computer Science',
        MULTISET[29,30],
        MULTISET( ROW( ... ) ) );

INSERT INTO Department
VALUES( 'Computer Science',
        MULTISET[28],
        MULTISET( SELECT ... FROM ... );

UPDATE Department
SET Buildings=Buildings MULTISET UNION MULTISET[17]
WHERE Name='Computer Science';
```

MULTISET type constructor (cont'd)

- **Unnesting** of a multiset

```
SELECT D.Name, Emp.LastName
FROM Department D,
     UNNEST( D.Employees ) Emp;
```

- **Nesting** using the COLLECT aggregation function

```
SELECT P.Name, COLLECT( S.hobby ) AS hobbies
FROM Person P NATURAL JOIN SpareTime S
GROUP BY P.Name;
```

PNF nested relations

An important subclass of nested relations

Principle

The Partitioned Normal Form (PNF) requires a **flat key** on every nesting level of a nested relation

PNF relation:

<table>
<tr><th rowspan="2">A</th><th colspan="2">D</th></tr>
<tr><th>B</th><th>C</th></tr>
<tr><td rowspan="2">1</td><td>2</td><td>3</td></tr>
<tr><td>4</td><td>2</td></tr>
<tr><td rowspan="2">2</td><td>1</td><td>1</td></tr>
<tr><td>4</td><td>1</td></tr>
<tr><td>3</td><td>1</td><td>1</td></tr>
</table>

Non-PNF relation:

<table>
<tr><th rowspan="2">A</th><th colspan="2">D</th></tr>
<tr><th>B</th><th>C</th></tr>
<tr><td rowspan="2">1</td><td>2</td><td>7</td></tr>
<tr><td>3</td><td>6</td></tr>
<tr><td>1</td><td>4</td><td>5</td></tr>
<tr><td>2</td><td>1</td><td>1</td></tr>
</table>

Partitioned Normal Form

Definition (PNF)

Let $R(X, Y)$ be a n-ary relation where X is the set of atomic attributes and Y is the set of relation-valued attributes; R is in partitioned normal form (PNF) iff

1. $X \rightarrow X, Y$ (X is a super-key)
2. Recursively, $\forall r \in Y$ and $\forall \mathtt{I}(r) \in \pi_r(R)$, $\mathtt{I}(r)$ is in PNF

- If $X = \emptyset$, then $\emptyset \longrightarrow Y$ must hold
- If $Y = \emptyset$, then $X \longrightarrow X$ holds trivially
  Thus a **1NF relation is in PNF**

Properties of PNF

1. A flat (1NF) relation is always in PNF
2. PNF relations are **closed** under unnesting
3. Nesting and unnesting operations **commute** for PNF relations
4. Size of PNF relations remains polynomial

Strong theoretical results and many practical applications

4NF counter-part of PNF

PNF relation and equivalent unnested relation

<table>
<tr><th rowspan="2">A</th><th colspan="2">E</th><th>F</th></tr>
<tr><th>B</th><th>C</th><th>D</th></tr>
<tr><td rowspan="2">1</td><td>2</td><td>3</td><td rowspan="2">1</td></tr>
<tr><td>4</td><td>2</td></tr>
<tr><td rowspan="2">2</td><td>1</td><td>1</td><td>2</td></tr>
<tr><td>4</td><td>1</td><td>3</td></tr>
<tr><td>3</td><td>1</td><td>1</td><td>2</td></tr>
</table>

$$\xrightarrow{\mu_E \circ \mu_F}$$

A	B	C	D
1	2	3	1
1	4	2	1
2	1	1	2
2	4	1	2
2	1	1	3
2	4	1	3
3	1	1	2

- $S = \mu_E \circ \mu_F(R)$ and $A \twoheadrightarrow BC$ holds in S
- PNF mimics 4NF for nested relations (A superkey)

PNF and MVD's and scheme tree

Preliminary statement

A **scheme tree** captures the logical structure of a nested relation schema and explicitly represents the **set of MVD's**

One more property of PNF relations

A nested relation R is in PNF iff the scheme of R follows a scheme tree with respect to the given set of MVD's

MVD's by example

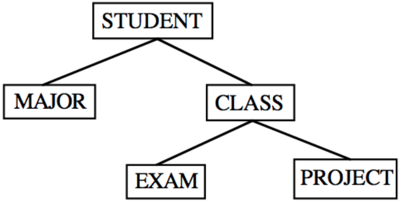
- Book db: $\{\text{Title} \twoheadrightarrow \text{Author}\}$
- Class db: Student, Major, Class, Exam, Project
  $\{\text{Student} \twoheadrightarrow \text{Major}, \text{Class} \twoheadrightarrow \text{Exam}\}$

Scheme—or schema—Tree

A tool for nested relation design

Definition (Scheme Tree)

A scheme tree is a tree containing at least one node and whose nodes are labelled with nonempty sets of attributes that form a disjoint partition of a set U of atomic attributes



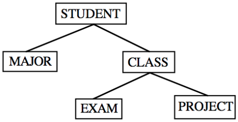
Design by MVD's

Pattern

Ancestors-and-self $\twoheadrightarrow$ Child-and-descendants

Example (cont'd)

- STUDENT $\twoheadrightarrow$ MAJOR
- STUDENT $\twoheadrightarrow$ CLASS EXAM PROJECT
- STUDENT CLASS $\twoheadrightarrow$ EXAM
- STUDENT CLASS $\twoheadrightarrow$ PROJECT



Nested Relation Schema

Definition (NRS)

A nested relation scheme (NRS) for a scheme tree T, denoted by $\mathcal{T}$, is a set defined recursively by:

1. If T is empty, i.e. T is defined over an empty set of attributes, then $\mathcal{T} = \emptyset$;
2. If T is a leaf node X, then $\mathcal{T} = \langle X \rangle$;
3. If A is the root of T and $T_1, \ldots, T_n$, $n \geq 1$, are the principal subtrees of T then $\mathcal{T} = \langle A, B_1 : \{\mathcal{T}_1\}, \ldots, B_n : \{\mathcal{T}_n\} \rangle$

Example (cont'd)

⟨STUDENT, Majors:{⟨MAJOR⟩},
Classes:{⟨CLASS, Exams:{⟨EXAM⟩}, Projects:{⟨PROJECT⟩} ⟩} ⟩

Nested relation over $\mathcal{T}$

Student	{Major}	{Class	{Exam}	{Project}}
Anna	Maths	CS100	mid-year	Project A
	Computing		final	Project B
				Project C
Bill		P100	final	Pract Test 1
	Physics			Prac Test 2
	Chemistry	CH200	test A	Exp 1
			test B	Exp 2
			test C	Exp 3

Exercises 1/2

1. Definitions

Nested relation, Nest, Unnest, Powerset, Flat-flat theorem, Scheme tree, NRS, PNF

2. True or False?

i) Nested algebra dominates RA.
ii) Nested relations is a restriction from NF^2.
iii) Nesting and unnesting are out of the scope of SQL.
iv) Nest is the right inverse of unnest for PNF relations.
v) Nesting a PNF relation produces a PNF relation.
vi) Any scheme tree encodes a set of MVD's.

Exercises 2/2

3. Nested relational queries ✎

1. Give result of each query example from Slide 6.
2. Give NRS from the following *serialized* schema tree:

$$(ab(c(de)(f))(gh(i(j))(kl))(m(n)))$$

4. Problem

Given a PNF relation R of sort $\langle A, B : \{\langle C, D\rangle\}\rangle$; (a) prove that the size of any instance $I(R)$ is bounded by a polynomial in adom$(I(R))$; (b) Show how to encode the same information within 2 flat tables.

Overview

Chapter eNF^2

1. Extending NF^2
2. Object Structure
3. Class Hierarchy

The eNF^2 data model

eNF^2 = Extended NF^2 Model

- Extend NF^2 model by introducing
  - various **type constructors** and
  - allowing their **free combination**
- Type constructors:
  - `set` $\{.\}$: create a set type of nested type
  - `tuple` $\langle . \rangle$: tuple type of nested type
  - `list` $(.)$: list type of nested type
  - `bag` $\{\!|.|\!\}$: bag—multi-set—type of nested type
  - `array` $[.]_n$: array type of nested type
  - `map` $[./.]$: key/value dictionary type of nested types
- First two are already available in RM and NF^2

The eNF^2 data model (cont'd)

The Evolution of Data Models

b.t.w. of sort comparison

- Relational Model $\quad \tau := \langle A_1 : \mathrm{dom}, \ldots, A_k : \mathrm{dom} \rangle$
- NF^2 $\quad \tau := \mathrm{dom} \mid \langle A_1 : \tau, \ldots, A_k : \tau \rangle \mid \{\tau\}$
- eNF^2

$$\tau := \mathrm{dom} \mid \langle A_1 : \tau, \ldots, A_k : \tau \rangle \mid \{\tau\} \mid (\tau) \mid [\tau]_n \mid \{\!|\tau|\!\} \mid [\tau/\tau]$$

Flavors by **restrictions**, such like nested relations for NF^2

Type constructors

- $\langle . \rangle$ $\{.\}$ $(.)$ $[.]_n$ $\{\!|.|\!\}$ $[./.]$ a.k.a. **Parametrizable Data Types**
- Construction based on '.', the **input data type**
- Define **own operations** for access and modification
- Similar to pre-defined parametrizable data types of programming languages
  - Generics in Java `java.util`
  - Templates in C++ STL
  - Type inference in OCaml

Comparison of type constructors

Type	Dupl.	Bounded	Order	Access by	Composite
Set $\{.\}$	✘	✘	✘	Iterator	✘
Bag $\{\!\vert . \vert\!\}$	✔	✘	✘	Iterator	✘
Map $[./.]$	✔	✘	✘	Key	✘
List $(.)$	✔	✘	✔	Position/Iter.	✘
Array $[.]_n$	✔	✔	✔	Index	✘
Tuple $\langle . \rangle$	✔	✔	✔	Name	✔

- All but tuple type constructors are **collection data types**
- Tuple type constructor is a **composite data type**

SQL ARRAY type constructor

- Introduced within SQL:1999

```
CREATE TABLE Contacts(
    Name          VARCHAR(40),
    PhoneNumbers  VARCHAR(20) ARRAY[4],
    Addresses     AddressType ARRAY[3] );
```

SQL ARRAY type constructor (cont'd)

- Array type constructor for record insertion
- Access to elements by **explicit position** $[k]$

```
INSERT INTO Contacts
VALUES( 'Doe',
        ARRAY['1234','5678'],
        ARRAY[ROW( '50 Otages', 'Nantes', '44000' )] );

UPDATE Contacts
  SET PhoneNumbers[3]='91011'
  WHERE Name='Doe';
```

SQL ARRAY type constructor (cont'd)

- Alternative access to element by **unnesting of collection**

```
SELECT Name, Tel.*
FROM Contacts,
     UNNEST( Contacts.PhoneNumbers ) WITH ORDINALITY
     AS Tel(Phone, Position)
WHERE Name='Doe';
```

- Further operations:
  - size `CARDINALITY()`
  - concatenation `||`

Object structure

- Complex value—state—conforms to object structure
- Type constructors are building blocks: tuple, set, list, array, bag, dictionary
- eNF^2 as the reference model
- Implementation within SQL3

Yet another popular restriction

- **Class** of sort τ following

$$\tau := \langle A_1 : \varrho, \ldots, A_k : \varrho \rangle$$
$$\varrho := \mathsf{dom} \mid \langle A_1 : \varrho, \ldots, A_k : \varrho \rangle \mid \{\varrho\} \mid (\varrho) \mid [\varrho]_n \mid \{\!|\varrho|\!\} \mid [\varrho/\varrho]$$

- Objects are **instances** of classes, a.k.a. class members
- An object o satisfies sort τ of its class c
- Essentially `struct` in C Programming Language

User-Defined Type in SQL3

UDT's occur at two levels:

- Columns of relations
- Tuples of relations

```
CREATE TYPE AddressType AS ( Street CHAR(50),
                             City   CHAR(50),
                             Zip    CHAR(5) );

CREATE TYPE BarType     AS ( Name   CHAR(20),
                             Addr   AddressType );

CREATE TABLE Bars OF BarType ( PRIMARY KEY (Name) );
```

Encapsulated object vs. row

- Bars is unary: tuples are objects with 2 components
- Grant **access privilege** to components
- **Type constructor**

```
INSERT INTO Bars
VALUES BarType( 'Le Flesselles',
                AddressType( '50 Otages',
                             'Nantes',
                             '44000' ) );
```

Encapsulated object vs. row (cont'd)

- **Observer** $A()$ and **Mutator** $A(v)$ for each attribute A
- Calls to implicit *getters* and *setters*, **redefinition** allowed

```
UPDATE Bars
SET Bars.Addr.Street('Allée Flesselles')
WHERE Bars.Name = 'Le Flesselles';

SELECT B.Name, B.Addr FROM Bars B;
```

Excerpt of the result set:
BarType('Le Flesselles',
AddressType ('Allée Flesselles', 'Nantes', '44000'))

Object behavior

Method := signature + body

Operation that apply to objects of a type

- $f(x)$ is invoked by sending a message to object o: $o.f(3)$
- Method
  - returns single value (may be a collection)
  - is typically written in general-purpose PL
  - could have unexpactable **side-effect**
- Implementation within ODL and SQL3

Disclaimer

Insight into object behavior is out of the scope of this series of slides

Corollary: main focus is the **structural part**

A word on eNF^2 in Oracle

- Supports **majority** of standard features as part of its object-relational extension—since 8i
  - **Multi-set** type constructor as `NESTED TABLE` type
  - **Array** type constructor as `VARRAY` type
  - **Object** (and **Tuple**) type constructor as `OBJECT` type
- Uses different syntax than ANSI/ISO SQL standard...

Alternative languages

Definition of object structures

• DDL part of SQL3	OR-Databases
• DDL part of [your favorite or-dbms]	OR-Databases
• Entity/Relationship (E/R) Model	Relational Databases
• Object Description Language (ODL)	OO-Databases
• Unified Modeling Language (UML)	OO-PL
• ...	

ODMG ODL

Example

```
class Bar {
  attribute string                              name;
  attribute struct addr {string street,
                         string city,
                         int    zip}    address;
  attribute enum lic {full, beer, none} license;
  attribute set< string >               drinks;
}
```

- Primitive types: int, real, char, string, bool, and *enumeration*
- Composite type: *structure*
- Collection types: set, array, bag, list, and dictionary

Class hierarchy

- Reuse of class definition
- A subclass is a refinement of its superclass

Definition (Class hierarchy)

A class hierarchy $(\mathcal{C}, \sigma, \prec)$ has 3 components:

1. a set $\mathcal{C}$ of class names
2. types τ's associated with these classes: $\sigma(c) = \tau$
3. specification of the is-a relationship $\prec$ between classes

Subtyping relationship

A subtype **inherits** value—and behavior—of a predefined type

Definition (Subtyping)

Let $(\mathcal{C}, \sigma, \prec)$ be a class hierarchy; subtyping relationship is the smallest partial order $\leq$ over types σ satisfying:

1. dom $\leq$ dom
2. if $\tau_i \leq \tau_i'$, $1 \leq i \leq n$, then $\langle A_1 : \tau_1, \ldots, A_n : \tau_n, \ldots, A_{n+k} : \tau_{n+k} \rangle \leq \langle A_1 : \tau_1', \ldots, A_n : \tau_n' \rangle$
3. if $\tau_1 \leq \tau_2$, then $\{\tau_1\} \leq \{\tau_2\}$, $(\tau_1) \leq (\tau_2)$, $\{\!|\tau_1|\!\} \leq \{\!|\tau_2|\!\}$ and $[\tau_1]_n \leq [\tau_2]_n$
4. if $\tau_1 \leq \tau_3$ and $\tau_2 \leq \tau_4$, then $[\tau_1/\tau_2] \leq [\tau_3/\tau_4]$
5. for each τ, $\tau \leq \text{ANY}$ (i.e. ANY is the top of the hierarchy)

Covariance and contravariance

Subtyping follows from wider to narrower: **covariance** only

Definition (Covariance)

Typing rules preserve the ordering on $\leq$

About the map type constructor

$$\begin{tikzcd}
A:\tau_1 \arrow[r, "{[a, b]}"] \arrow[dr, dashed] & B:\tau_2 \\
C:\tau_3 \arrow[u] \arrow[ur, dashed] \arrow[r, dashed] & D:\tau_4 \arrow[u]
\end{tikzcd}$$

- **Contravariance** reverses the ordering: type safety in PL

Well-formed class hierarchy

Property

A class hierarchy $(\mathcal{C}, \sigma, \prec)$ is **well-formed** iff for each pair c_1, c_2 of classes, $c_1 \prec c_2$ implies $\sigma(c_1) \leq \sigma(c_2)$

Example

- $\sigma(\text{Person}) = \langle \text{name} \rangle$
- $\sigma(\text{Teacher}) = \langle \text{name}, \text{dpts} : \{\langle \text{id} \rangle\} \rangle$
- $\sigma(\text{Student}) = \langle \text{name}, \text{major}, \text{enrol} : \{\text{dom}\} \rangle$
- $\sigma(\text{Lecturer}) = \langle \text{name}, \text{dpts} : \{\langle \text{id}, \text{office} \rangle\}, \text{contacts} : [\text{dom}]_3 \rangle$
- $\sigma(\text{Tutor}) = \langle \text{name}, \text{dpts} : \{\langle \text{id} \rangle\}, \text{labs} : \{\!|\langle \text{day}, \text{room} \rangle|\!\} \rangle$

$\{$Student, Teacher$\} \prec$ Person, and $\{$Lecturer, Tutor$\} \prec$ Teacher

Subtyping within SQL

`UNDER` clause with `NOT FINAL` statement in the base type

```
CREATE TYPE PersonType AS (
    Name        VARCHAR(20)   NOT NULL,
    DateOfBirth DATE,
    Gender      CHAR)
NOT FINAL;

CREATE TYPE StudentType UNDER PersonType AS (
    StudentID   VARCHAR(10),
    Major       VARCHAR(20)
);

CREATE TABLE Student OF StudentType;
```

Multiple Inheritance

- **More than one superclass** vs. single inheritance
- The is-a relationship forms a directed acyclic graph (DAG)
- A subclass inherits state and behavior from **all** its superclasses
- Potential for ambiguity, e.g., fields with the same name
- The "*diamond problem*": $D \prec \{B, C\} \prec A$
- SQL does not support multiple inheritance of UDT's

Inheritance within ODL

```
class Person {
    attribute string    name;
    attribute character gender; }

class Teacher extends Person {...}
class Student extends Person {...}

class TeachingFellow extends Teacher, Student {
    attribute string    degree; }
```

- How many names and genders for a single TF ?!

Membership in a class hierarchy

Definition (Object assignment)

A function π mapping each name in $\mathcal{C}$ to a finite set of objets

- Proper extension of c: $\pi(c)$
- Set of database objects: $O = \{\pi(c) \mid c \in \mathcal{C}\}$
- **Extension** of c: $\pi^*(c) = \bigcup\{\pi(x) \mid x \in \mathcal{C} \ \wedge \ x \prec c\}$
- $\pi^*(c_1) \subseteq \pi^*(c_2)$ whenever $c_1 \prec c_2$

Definition (Substitutability principle)

Value of type S can be sustituted to value of its supertype T

Alternative memberships

Properties

- **Complete** assignment: $\pi^*(c) \neq \emptyset \to \pi(c) = \emptyset$
- **Disjoint** assignment: $\pi(c_1) \cap \pi(c_2) \neq \emptyset \to (c_1 \prec c_2 \vee c_2 \prec c_1)$

Each class may have direct subclasses with:

- complete vs. partial assignment
- disjoint vs. overlapping assignment

Extension in ODL

- **Extent** declaration: *named* set of objects of the same type
  - Class ∼ Schema of a relation
  - Extent ∼ Instance of a relation
- Optional **Key** declaration: unicity constraint

```
class Course ( extent Courses
               keys   id, (dept, title), (classroom, time) )
{...};

SELECT c.id, c.title FROM Courses c
WHERE c.dept='Computer Science';
```

- Object Query Language (OQL): SQL-like for pure object db's
- Alias for extent (c) is mandatory: typical class member

"Subtabling" within SQL

No native extension for types in SQL: create table for each UDT

Table inheritance!

```
CREATE TABLE Person OF PersonType;
CREATE TABLE Student OF StudentType UNDER Person;
```

- A `Person` row matches at most one `Student` row
- A `Student` row matches exactly one `Person` row
- Inherited columns are inserted only into `Person` table
- Delete `Student` row deletes matching `Person` row

"Subtabling" within SQL (cont'd)

- Default: retrieve the extension $\pi^*(\text{Person})$ with all subtable rows

```
SELECT P.Name FROM Person P;
```

- `ONLY` clause: retrieve the proper extension $\pi(\text{Person})$

```
SELECT P.Name FROM ONLY (Person) P;
```

Open issues

Multiple-table inheritance ? Propagation of referential integrity constraints ? Index ? . . .

Basics of relational mapping

- Classes are all distinct tables
- Keys must be defined
- The three ways to cope with class hierarchy:
  1. E/R-style: one partial table by subclass with key+specific fields
  2. OO-style: one full table by subclass
  3. Null-style: all subclasses embedded within one single base table

Example

Person(<u>name</u>, gender)
Teacher(<u>name</u>, dpt)
Student(<u>name</u>, major)

Person(<u>name</u>, gender)
Teacher(<u>name</u>, gender, dpt)
Student(<u>name</u>, gender, major)

Person(<u>name</u>, gender, dpt, major)

Exercises 1/2

1. Definitions

eNF^2, Class, Class member, UDT, Observer, Mutator, Method, ODL, Class hierarchy, Subtyping, Covariance, Diamond problem, Substitutability principle, Extension, OQL

2. True or False?

i) eNF^2 dominates NF^2.

ii) Map is a composite type constructor.

iii) SQL implements eNF^2.

iv) Subtyping rules are covariant only.

v) $\pi(c) = \emptyset$ except for the leaves, is a full complete assignment.

vi) SQL Subtabling implements relational mapping in E/R-style.

Exercises 2/2

3. Misc

1. Exhibit a class hierarchy that is not well formed.
2. Give in ODL the `Person` class hierarchy of the example slide.

4. Problem

How many relations are required, using the OO-style mapping, if there is a 3-level hierarchy with out-degree 4, and that hierarchy is: (a) disjoint and complete at each level, (b) disjoint and partial at each level, and (c) overlapping and partial.

Overview

Chapter ID's and Refs

1. OID's and References
2. Relationships
3. Graph Databases

Object identity

- Persistent objects are given an **Object IDentifier** (OID)
- Used to manage *inter-object references*
- OID's are
  - **unique** among the set of objects stored in the DB
  - **immutable** even on update of the object value
  - **permanent** all along the object lifecycle
- OID's are not based on physical representation/storage of object (i.e., $\neq$ ROWID or TID, $\neq$ @object)

Ultimate object representation

Definition (Object)

An object is a pair (o, ϑ), with o being the OID and ϑ is the value

- Object identity is given by the OID
- Object value is not required to be unique

Values by example

- In the *class*-oriented restriction of eNF^2, values ϑ are
  - tuple-based complex values:
    $(o_1, \langle \text{title} : \text{'cs123'}, \text{desc} : \text{'...'} \rangle)$
    $(o_2, \langle \text{title} : \text{'cs987'}, \text{desc} : \text{'...'} \rangle)$
    $(o_3, \langle \text{name} : \text{'Doe'}, \text{major} : \text{'cs'}, \text{year} : \text{'junior'}, \text{enrol} : \{o_1, o_2\} \rangle)$
  - OID to achieve aliasing: (o_4, o_3)
  - *nil* for nullable reference: (o_5, nil)

Composition graph

Structural representation of an object as a labeled directed graph

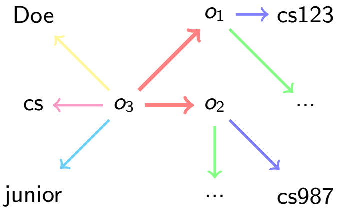
$$\text{struct}(o) := G(V, E)$$

where

- Vertices $V \subset O \cup \text{dom}$ are OID's and atomic values
- Edges $E \subseteq V \times \mathcal{A} \times V$ are labeled with symbols from $\mathcal{A}$, the set of field names
- Draw an edge (o_i, x) whenever $x \in \{o_j, a\}$ occurs in the value of o_i, a being an atomic value in dom

Composition graph (cont'd)

Example for object o_3



name major year
title desc enrol

Extend to a—cyclic—graph: *teacher* $\rightarrow$ *dpt* $\rightarrow$ *employees*

Statement

Object db is essentially **a huge persistent relational graph**

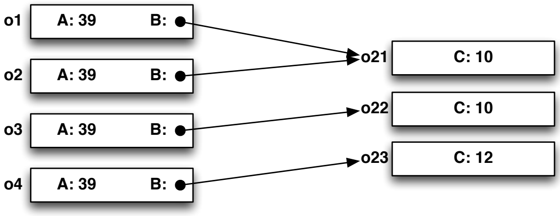
Equality

- **Identity** ($==$) is checked by means of OID's comparison
- Composition graph allows to compare two objects for equality
  - **Shallow equality** ($=$): graphs must be identical, including OID's
  - **Deep equality** ($=_*$): isomorphic graphs with different OID's but atomic values are equal

Properties

$$
\begin{aligned}
o_i == o_j \quad &\longrightarrow \quad o_i = o_j \\
o_i = o_j \quad &\longrightarrow \quad o_i =_* o_j
\end{aligned}
$$

Equality (cont'd)



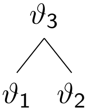
True	False
$o_1.B == o_2.B$	$o_1 == o_2$
$o_1 = o_2$	$o_1 = o_3$
$o_1 =_* o_3$	$o_1 =_* o_4$
$o_{21} = o_{22}$	$o_{21} == o_{22}$

Object expansion

Definition (Expansion)

Expansion of an object o, denoted expand(o), is the—possibly infinite—tree obtained by replacing each object by its value recursively

Example of expand(o_3)



- Infinite expansion: cycle in the composition graph
- Deep equality can be checked from expansion traversal

Object persistence

- In OO-PL, objects are **transient**
- Persistence is orthogonal to object types—classes
- Many policies to come up with **persistent** objects:
  - Object creation and explicit declaration
  - Homogeneous collection by **extension**
  - **Reachability**: declare persistent objects by name, and the system makes persistent all reachable objects at any level of the composition graph

Reachability vs. extension

Names

1. Extension: all class members are designated by the name of a single collection
2. Reachability: objects are linked to the name of their **root of persistence**

On delete

1. Extension: must detect an orphan (?) and retrieve the object from its collection name
2. Reachability: garbage collecting when an object has a null in-degree

Types revisited

The family of *types* τ over the set $\mathcal{C}$ of class names is as follows:

$$\tau := \langle A_1 : \varrho, \ldots, A_k : \varrho \rangle$$
$$\varrho := \mathsf{dom} \mid c \mid \langle A_1 : \varrho, \ldots, A_k : \varrho \rangle \mid \{\varrho\} \mid (\varrho) \mid [\varrho]_n \mid \{\!|\varrho|\!\} \mid [\varrho/\varrho]$$

- dom may be refined into primitive types of the language
- c is any class name in $\mathcal{C}$

Types revisited (cont')

- Object assignment π is basically OID assignment
- $\textsc{any} \in \mathcal{C}$ is a—singular—type such that $\text{dom}(\textsc{any}) = [\![O]\!]$
- Subtyping relationship is extended to:
  6. if $c_1 \prec c_2$, then $c_1 \leq c_2$
- Semantics of a class c is $\text{dom}(c) = \pi^*(c) \cup \{nil\}$

SQL3 References

Principle

If τ is a type, then $\mathtt{REF}(\tau)$ is a **type of references** to τ

- Weak translation of OID's into SQL world
- Unlike OID's, a `REF` is **visible** although it is gibberish

```
CREATE TYPE SellType AS (
  bar    REF(BarType)  SCOPE Bar,
  beer   REF(BeerType) SCOPE Beer,
  price FLOAT );
```

Following REF's and dereferencing

```
CREATE TABLE Sell OF SellType (
  REF IS sellID SYSTEM GENERATED,
  PRIMARY KEY (bar, beer) );

SELECT DEREF(s.beer) AS beer
FROM Sells s
WHERE s.bar->name = 'Le Flesselles';
```

- It would have required a join or nested query otherwise

Relationships

- Operate at the type system—class definition—level
- Connect entities/classes/types one with each other
- Binary relationships as **partial multi-valued functions**
- Decide for a direction: `contains` or `isIncluded` or both
- Prevent from redundancy: computed relationships
- Multiway relationships simulated by *connecting* classes

ODL example

```
class Sell {
  attribute    real price;
  relationship Bar  theBar;
  relationship Beer theBeer;
}
```

Multiplicity of relationships

- For binary relationships
  - One-one: `class/class`
  - Many-one: `set`[2]`/class`
  - Many-many: `set/set`
- 'One' means **at most one**
- Many-one variants: **aggregation** and **composition** (1..1)
- Weak class: id depends on master class id's ($\neq$ OID's)

```
class Employee (key (name, affiliated_to)) {  // weak class
  attribute    string        name;
  relationship Department    affiliated_to;   // one-one
  relationship set<Task>     assigned_to;     // many-one
  relationship list<Project> participates_in; // many-many
}
```

[2] or any collection type constructor.

Basic properties of partial multi-valued functions

- A function $f : X \to Y$ can be
  - **total**: domain of f is X
  - **injective**: for all a,b in domain of f, if $f(a) = f(b)$ then $a = b$
  - **surjective**: range (or codomain) of f is Y
  - **bijective**: both injective and surjective
- The **inverse** function $f^{-1} : Y \to X$ satisfies $f^{-1} \circ f = \mathrm{Id}$, with $\circ$ extended to sets of values

Comments

- Impact on design choices and further encoding
- Problems arise especially when mixed with inheritance

Implementation of the inverse function

- Two relationships are right inverses by means of
  - Both ways of the same link in graphical languages
  - `inverse` statement in ODL
  - Not supported in SQL3

Example in ODL

```
class Employee { ...
  relationship list<Project> participates_in
               inverse       Project::members; }
class Project { ...
  relationship set<Employee> members
               inverse       Employee::participates_in; }
```

Relational mapping

- Relationships are essentially all distinct tables
- Key fields both parts come into play
- Exceptions:
  - Supporting relationship of a weak class does not require a separate table
  - Inlining of aggregations and compositions
- Discussion about inlining each *-one relationship
- Implement relationships one way only

OQL features

- Query can include **path expressions** rather than joins:

```
SELECT s.beer.name, s.price
FROM Sell
WHERE s.bar.name='Le Flesselles';
```

- Alternative query

```
SELECT s.beer.name, s.price
FROM Bar b, b.beerSold s
WHERE b.name='Le Flesselles';
```

- Collections cannot be further extended by dot notation
- Collections can be part of the FROM clause

OQL features (cont'd)

- Result type is basically $\{\!|\langle . \rangle|\!\}$
- Complex result type can be constructed in query

```
SELECT DISTINCT struct( e.name,
                        projects:(
    SELECT p.projectId
    FROM e.participates_in AS p) )
FROM Employees AS e;
```

- Result type:

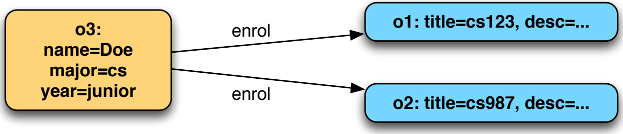
$$\{\langle \text{name} : \text{string}, \text{projects} : \{\!|\text{int}|\!\}\rangle\}$$

Relationship as first-class citizen

Reminder

- Object db as a (multi-)relational graph
- Can be further simplified to a vertex-attributed relational graph where atomic key/value pairs are an extended part of the object node itself

Example of o_3



Claim

Object db's are graph db's

Requirements for graph databases

The 3D graph data model [Angles et al., ACM CS 2008]

1. Data structure
   - data and schema are (separate) **graphs**
   - standard abstractions: is-a is-type-of is-part-of is-composed-by is-member-of is-associated-to
2. Update and query language
   - graph transformations
   - primitives on paths, neighborhoods, subgraphs, graph patterns, connectivity and graph statistics (diameter, centrality, etc.)
   - multi-relational graph algorithms
3. Integrity constraints
   - schema-instance consistency, identity, referential integrity and functional and inclusion dependencies

Requirements for graph databases (cont'd)

Definition (Graph database (tentative of))

Any storage system that provides index-free adjacency

- Each vertex has direct references to its adjacent vertices
  - act as a **mini-index**
- $\mathcal{O}(1)$ to move from a vertex to its neighbors
- $\mathcal{O}(\log n)$ b.t.w. of an index in non-graph db's

Dominant and alternative models of graph db's

Data model

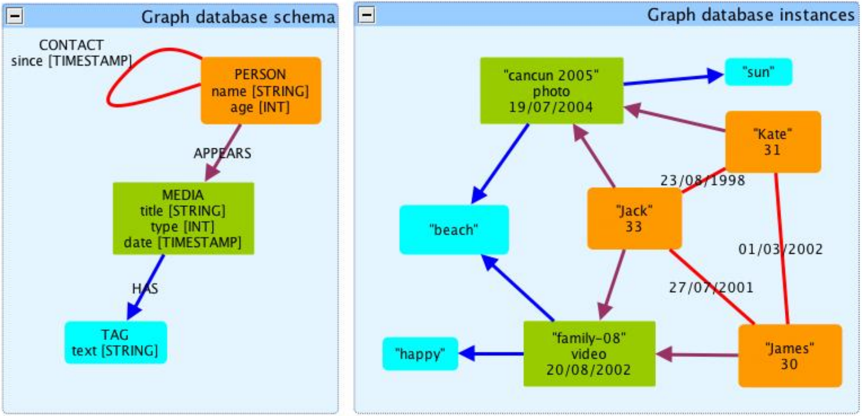
An (attributed) multi-relational labeled digraph $G(V, E)$ where

- V is a finite set of distinct node labels (id's)
- $E \subseteq V \times \Sigma \times V$, Σ being a finite alphabet of directed edge—**relationship**—symbols
- optional attributes—**properties**—may apply to both vertices and edges as a set of key/value pairs

Extension to

- hypernode: nested graphs where nodes are graphs themselves
- hypergraph: with hyperedges, i.e., sets of nodes
- multigraph: multiple single-relational edges

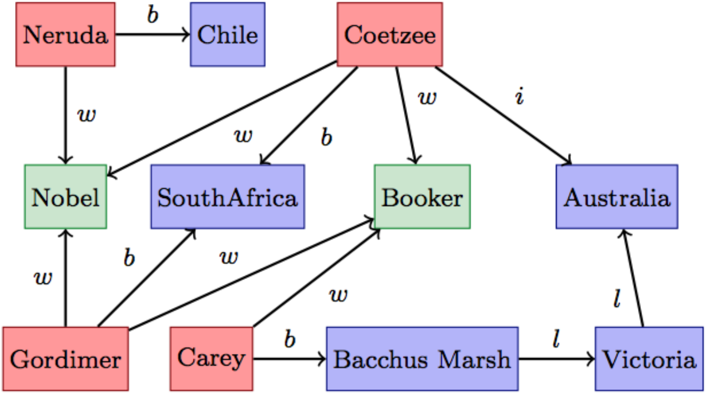
Graph db example



[3]

[3] Source: DEX Javadoc

Another graph db example



[4]

b:bornIn *w*:hasWon *i*:livesIn *l*:locatedIn

[4] Source: (P.T. Wood, SIGMOD Record 2012)

Conjunctive Queries

Implements **subgraph matching**

Definition (Graph CQ's)

$$\mathrm{ans}(\vec{z}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, a_i, y_i)$$

where x_i, y_i are node variables or constants, each z_j is x_i, y_i or any constant, and $a_i \in \Sigma$

Example

$$\begin{aligned} \mathrm{ans}(x) \leftarrow & (x, \mathrm{hasWon}, \mathrm{Nobel}), (x, \mathrm{hasWon}, \mathrm{Booker}), \\ & (x, \mathrm{bornIn}, \mathrm{SouthAfrica}) \end{aligned}$$

Conjunctive Regular Path Queries

Implements **reachability** by path expressions

Definition (Graph CRPQ's)

$$\text{ans}(\vec{z}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, r_i, y_i)$$

Extend CQ's to r_i as a **regular expression** over Σ

Example

$$\begin{aligned} \text{ans}(x) \quad & \leftarrow (x, \text{hasWon}, \text{Booker}), (x, r, \text{Australia}) \\ r \quad & := \text{citizenOf} \mid ((\text{bornIn} \mid \text{livesIn}).\text{locatedIn}^*) \end{aligned}$$

Extended CRPQ's

Paths may

- occur in the output by means of **free path variables**
- be compared within a **regular relation**

Examples

Retrieve every path between nodes r and s that go through node e:

$$\text{ans}(\pi_1, \pi_2) \leftarrow (r, \pi_1, e), (e, \pi_2, s)$$

Retrieve all pairs (x, y) connected by paths following pattern $a^n.b^n$:

$$\text{ans}(x, y) \leftarrow (x, \pi_1, z), (z, \pi_2, y), a^*(\pi_1), b^*(\pi_2), (\bigcup_{a,b\in\Sigma} (a, b))^*(\pi_1, \pi_2)$$

Extended CRPQ's (cont'd)

Definition (Graph ECRPQ's)

$$\mathrm{ans}(\vec{z}, \vec{\chi}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq p} R_j^{k_j}(\vec{\omega_j})$$

where χ_ℓ, π_i, ω_{jt} are path variables, and $R_j^{k_j}$ is a regular expression that defines a regular relation over Σ

Extension to approximate matching and ranking

- Define an **edit distance** $d_e(x, y)$ on Σ^*
- Operate with a regular expression over triples of the form (a, k, b), $a, b \in \Sigma \cup \{\epsilon\}$, k the cost of substitution

Aggregation and arithmetic predicates

count sum min max $+ - * /$ for computing:

- degree, eccentricity of a node
- distance between two nodes
- diameter of the graph
- *etc.*

Example

Length of the *shortest path* between each pair of nodes

$$
\begin{array}{lcl}
\mathrm{len}(x,x,x,0) & \leftarrow & \mathrm{dist}(x,y,\ell) \\
\mathrm{len}(x,x,x,0) & \leftarrow & \mathrm{dist}(y,x,\ell) \\
\mathrm{len}(x,z,y,d) & \leftarrow & \mathrm{sp}(x,z,s), \mathrm{dist}(z,y,\ell), d = s + \ell \\
\mathrm{sp}(x,y,\min(d)) & \leftarrow & \mathrm{len}(x,z,y,d)
\end{array}
$$

Node creation

Skolem function

- remove existential quantifiers within FO formula: basically, $\exists x.P(x)$ becomes $P(a)$ with a a constant
- substitute to free variables to create new nodes in queries

Example

From a people-centric network to a city-centric network

$$
\begin{array}{lcl}
\text{ans}(f(c), \text{is-a}, \text{city}) & \leftarrow & \Delta(p, c) \\
\text{ans}(f(c), \text{name}, c) & \leftarrow & \Delta(p, c) \\
\text{ans}(f(c), \text{population}, \text{count}(p)) & \leftarrow & \Delta(p, c) \\
\Delta(p, c) & \leftarrow & (p, \text{is-a}, \text{person}), (p, \text{livesIn}, c)
\end{array}
$$

The Web of Data

- Resource Description Framework (RDF) for the Semantic Web
- RDF statement: (subject, predicate, object)
- **Triple store** is a restriction of graph db

RDF Data Model

Vertex set is split into URIs (U), literals (L), and blank/anonymous nodes (B), such that:

$$G \subseteq ((U \times B) \times U \times (U \times B \times L))$$

- Extension to **named graphs** as $\mathrm{ng} = (n, g)$ with $n \in U$ and $g \in \mathcal{G}$, the family of RDF graphs

The Web of Data (cont'd)

SPARQL

Graph pattern-based SQL-like query language for RDF triple stores

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

Semi-structured data

eXtended Markup Language (XML)

- Tree-like structures: **rooted digraphs**!
- Labeled unbounded ordered trees: restricted type of graph
- Referencing mechanism: simulation of arbitrary graphs
- Self-describing: hierarchical structure within the document
- Implicit *composition* relation only

XML query languages

- **XPath**: path expressions
- **XQuery**: SQL-like query language for XML db's

Object db's vs. Graph db's

At a glance:

- Structural part is essentially the same
- Querying capabilities may converge

Main differences:

- Graph db's are schema-less
- Object db's equip objects with operations

Small variants:

- Graph db's handle properties on edges
- Graph db's emphasize interconnections and their properties
- Object db's emphasize dynamic state and behavior of objects

Exercises 1/2

1. Definitions

OID, Composition graph, Shallow equality, Deep equality, Expansion, Persistence by reachability, Partial multi-valued function, Graph database, Graph CQ, Graph CRPQ, Graph ECRPQ, RDF data model

2. True or False?

i) OID's are kind of primary keys.
ii) Given $=$ and $=_*$ are resp. $=_0$ and $=_\infty$, then $=_{k+1}$ refines $=_k$.
iii) One-one relationships are injective functions.
iv) Graph db's cannot implement class hierarchies.
v) Subgraph matching is basic requirement for graph db querying.
vi) RDF triple stores are graph db's.

Exercises 2/2

3. ODL relationships ✎

1. Give an ODL design with all the inverse relationships for the `Person` class to represent a genealogy.
2. What makes a relationships its own inverse ?

4. Problem

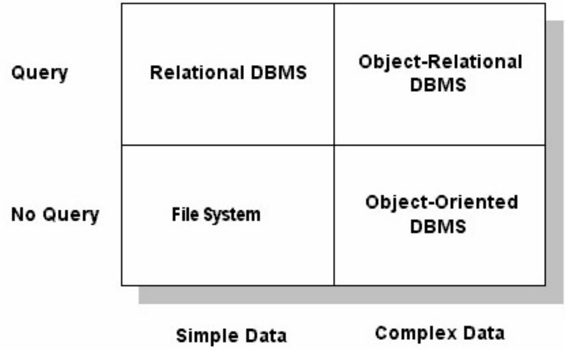
Prove that expand(o) is a regular tree, i.e., it has a finite number of distinct subtrees.

From Lineland to Spaceland

Object-Oriented paradigm brings to the—relational—data world

- Mashup of:
  1. Databases
  2. OO Programming Languages
  3. Conceptual/Semantic Modeling
- Practical approaches to contemporary issues
- Lack of strong mathematical foundations

The Matrix



M. Stonebraker: *Object-Relational DBMS: The Next Great Wave*, MK, 1998

15 years later, OO-DBMS in South-East quadrant is questionable

Impedance Mismatch revisited

Find a sunset picture taken within a coastal zone by a professional photographer

```
SELECT p.id
FROM slides p, area a, a.landmarks l
WHERE sunset (p.picture) AND
      p.owner.occupation = 'photographer' AND
      a.type = 'coastal' AND
      contains (p.caption, l.name) ;
```

- User-defined functions: `sunset()` `contains()`
- Path expression: `P.owner.occupation`
- Collection as table: `area.landmarks`

The First Manifesto

M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. **The Object-Oriented Database System Manifesto**. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, pages 223-240, Kyoto, Japan, December 1989

13 must-have features of OO-DBMS

- 8 from **Object-Oriented Programming Languages**
  complex objects, object identity, encapsulation, types and classes, inheritance, polymorphism, completeness, extensibility
- 5 from **Databases**
  persistence, secondary storage management, concurrency, recovery, *ad hoc* query facility

ODMG Standard

- Object Database Management Group
  - **1991** ODMG was created by R. Cattell of Sun Microsystems
  - **2000** Latest standard: ODMG 3.0
  - **2001** ODMG disbanded to focus on Java Data Object (JDO)
  - **2006** OMG Object Database Technology (ODBT) Working Group for the 4th generation of OO-DBMS standard
- Four components
  1. Object Model
  2. Object Definition Language (ODL)
  3. Object Query Language (OQL)
  4. Language Binding for C++, Java, Smalltalk

Object Query Language (OQL)

- Extension of the SQL-92 standard: object-oriented notions, like complex objects, object identity, path expressions, operation invocation etc.
- High level constructs to deal with sets of objects and primitives for structures, list, arrays etc.
- Functional language where operators can freely be composed, as long as the operands respect the type system
- OQL does not provide explicit **update** operators but rather invokes operations defined on objects for that purpose

OO-DBMS vs. OR-DBMS vs. O/R Mapping

Relation as first-class citizen?

- Yes: SQL3
  - PostgreSQL, IBM DB2, Oracle, Microsoft SQL Server, Sybase
- No: ODMG ODL+OQL
  - db4o, Versant, ObjectStore, ObjectDB, Native Queries, LINQ
- Don't care: PL coupled with (R-)DBMS Mapping Framework
  - Hibernate, JPA, JDO, CodeIgniter, Symfony, Django, EF