Functional Dependencies

Databases

Ira Assent

ira@cs.au.dk

Data-Intensive Systems group, Department of Computer Science, Aarhus University, DK

Intended learning outcomes

- Be able to
  - (leftover) schema modifications in SQL
  - Identify anomalies and how they relate to functional dependencies
  - Work with Functional Dependencies

ira@cs.au.dk

Recap: Complex SQL

`SELECT` *desired attributes* `FROM` *one or more tables*

`WHERE` *condition about the involved rows*

Subqueries

Non-correlated – evaluated once

```
SELECT Lname FROM EMPLOYEE WHERE Salary > (SELECT
  AVG(Salary) FROM EMPLOYEE);
```

Correlated – evaluated per value / tuple in the outer query

```
SELECT E.Lname FROM Employee E WHERE salary >(SELECT
  AVG(salary)FROM Employee WHERE Dno = E.Dno));
```

`IN, ANY, ALL, EXISTS, NOT, UNIQUE,` comparisons, aggregates, functions, …

`NULL` – unknown / non-existent: 3 truth values; result only TRUE conditions

`JOIN: NATURAL, EQUIJOIN, INNER JOIN, OUTER JOIN, LEFT JOIN, RIGHT JOIN`

https://dev.mysql.com/doc/refman/8.0/en/select.html

ira@cs.au.dk

Recap: `GROUP BY` and `Having`

- `GROUP BY` partition relation into subsets of tuples
  - Based on grouping attribute(s)
- HAVING clause specifies conditions for groups (otherwise, group not in result)
  - Attributes in HAVING must be aggregates or mentioned in GROUP BY (or functionally dependent, we'll get to that…)

```
SELECT study, semester AVG(grade)FROM Grades GROUP BY
study, semester HAVING AVG(grade)> 10;
```

  - For each distinct combination of attributes study and semester what is the average grade
  - Having condition includes only groups where the average is above 10
    - A group with an average of say 8.2 is simply not returned
      - E.g. if DS students in Spring 2024 have an average grade of exactly 10, then not part of result

study	semester	AVG (grade)
CS	Fall 2024	10.2
CS	Spring 2024	10.1
DS	Fall 2024	10.5

ira@cs.au.dk

Which works?

1. `SELECT shop FROM Sells GROUP BY song HAVING COUNT(song)<2;`
2. `SELECT shop, MAX(song) FROM Sells WHERE COUNT(song)<2;`
3. `SELECT shop, COUNT(song), AVG(price) FROM Sells GROUP BY shop HAVING COUNT(song)<2;`
4. `SELECT shop, AVG(song) FROM Sells GROUP BY shop HAVING COUNT(song)<2;`

Sells

shop	song	price
PearMusic	Feather birds	1
Hotify	Feather birds	2
PearMusic	Alone	1
UTooba	Not like me	.5
PearMusic	Not like me	.7

ira@cs.au.dk

High quality databases – Version 2.0

- Good E/R design model generally leads to good design
  - What is "generally", what is "good"?
- We will now identify quality issues and their causes
  - "anomalies" and the guidelines they violate
- We will then move to a formal description of the basic building blocks in analyzing and normalizing tables
  - "functional dependencies"
  - Use these to define normal forms, and to transform databases into normal forms

ira@cs.au.dk

Guidelines for good design

- One table – one meaning
  - Design relation schema so that it is easy to explain its meaning
  - Do not combine attributes from multiple entity types and relationship types into single relation

Guideline 1

- Avoid anomalies; else mark and handle in program
  - Design base relation schemas so that no anomalies in relations
  - If any anomalies: note them clearly, make sure that the programs that update the database will operate correctly

Guideline 2

- Often caused by base relations (Meeting, People) mixed in one table
- Example: information about owners is duplicated

Arrangements

meetid	topic	date	userid	name	group	office
34716	dDB	2023-08-28	ira	Ira Assent	vip	Ny-357
34717	dDB	2024-01-23	ira	Ira Assent	vip	Ny-357
42835	TA meeting	2023-08-18	aas	Annika Schmidt	phd	NULL

ira@cs.au.dk

NULL Values in Tuples

- May group many attributes together into a **fat** relation
  - Can end up with many NULLs
- Problems with NULLs
  - Wasted storage space
  - Problems understanding meaning
  - And needs to be taken into account when querying!

id	name	course	office	phone	fax	certificate
1	Eve	DB	Ny-343	NULL	NULL	SciTeach
2	Chris	DB	NULL	NULL	NULL	NULL
2	Tom	Prog	NULL	1234-9999	NULL	NULL

- Avoid placing attributes in a base relation whose values may frequently be NULL
- If NULLs are unavoidable:
  - Make sure that they apply in exceptional cases only, not to a majority of tuples

Guideline 3

ira@cs.au.dk

Generation of Spurious Tuples

- Poor decomposition of relation

meetid	topic	date	userid	group
34716	dDB	2023-08-28	ira	vip
34717	dDB	2024-01-23	ira	vip
42835	*Logic*	*2023-08-18*	*fra*	*vip*

name	group	office
Annika Schmidt	phd	NULL
Ira Assent	*vip*	*Ny-357*
Frank Roehr	vip	Ho-010

  - Attribute group connecting the two tables is not key
    - so combining with e.g. NATURAL JOIN generates **spurious tuples** that do not exist in original data

meetid	topic	date	userid	name	group	office
34716	dDB	2023-08-28	ira	Ira Assent	vip	Ny-357
42835	Logic	2023-08-18	fra	Ira Assent	vip	Ny-357
34717	dDB	2024-01-23	ira	Ira Assent	vip	Ny-357
42835	Logic	2023-08-18	fra	Frank Roehr	vip	Ho-010

Avoiding Spurious Tuples

- Ensure relationships between tables make sense
  - Referential integrity constraints (foreign key)
  - Check for ambiguous attribute names in semantically related tables
    - Recall NATURAL JOIN matches on identical attributes: e.g. Student (name,…) Course(name,…) could potentially lead to matches on names of same type, but with different semantics

- Design relation schemas to be joined with equality conditions on attributes that are appropriately related
- Guarantees that no spurious tuples are generated
- Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations

Guideline 4

ira@cs.au.dk

Insertion anomalies

- Two types of insertion anomalies

1. If we want to insert a tuple on some meeting into Arrangements, must either include (consistent) attribute values for user as well or NULLs if the user information is not known
   - E.g. creating two meetings for ira, they need to agree on office
2. It is difficult to insert user who does not have any meetings
   - Would require entering NULL values
     - Also for the key meetid, which really cannot be NULL

Arrangements

meetid	topic	date	userid	name	group	office
34716	dDB	2023-08-28	ira	Ira Assent	vip	Ny-357
34717	dDB	2024-01-23	ira	Ira Assent	vip	Ny-357
NULL	NULL	NULL	aas	Annika Schmidt	phd	NULL

ira@cs.au.dk

Modification Anomaly

- Updating a row may cause inconsistent information unless all redundant values are updated as well
  - E.g. update office information regarding meeting 34717

Arrangements

meetid	topic	date	userid	name	group	office
34716	dDB	2023-08-28	ira	Ira Assent	vip	Ny-357
34717	dDB	2024-01-23	ira	Ira Assent	vip	Ny-343
42835	TA meeting	2023-08-18	aas	Annika Schmidt	phd	NULL

- Ira now has conflicting office information

ira@cs.au.dk

Deletion Anomaly

- Deletion anomaly
  - deleting a row causes too much information to disappear

Arrangements

meetid	topic	date	userid	name	group	office
34716	dDB	2023-08-28	ira	Ira Assent	vip	Ny-357
34717	dDB	2024-01-23	ira	Ira Assent	vip	Ny-357
~~42835~~	~~TA meeting~~	~~2023-08-18~~	~~aas~~	~~Annika Schmidt~~	~~phd~~	~~NULL~~

- When Annikas meeting is deleted, we no longer know anything about her

ira@cs.au.dk

Fixing issues: dependencies to the rescue

- Want to detect possible sources of anomalies in relations
- Functional dependencies are a formal tool for doing that
- **Functional dependency** $X \rightarrow Y$ is constraint on possible tuples forming relation state r of R

  between sets of attributes $X, Y \subseteq R$ such that

  $t_1[X] = t_2[X] \rightarrow t_1[Y] = t_2[Y]$

- If value of one set is known, then the other is as well
  - E.g. value of zip implies value of city: 8000 → Aarhus
- This is a property of the miniworld, not of the database
- So, like all the other constraints – originate from miniworld!
  - Depending on where in the world you are (read: which geography your database handles) different answer

ira@cs.au.dk

Functional Dependencies

- Functional dependency (FD) in a table: $a_1,...,a_n \rightarrow b$
  - Implies that values of attributes $a_1,...,a_n$ (**left hand side**) determine value of attribute b (**right hand side**) in any row
  - E.g. name → group or meetid → date
- Generalizes notion of primary key, but primary key is always *minimal* (smallest subset) and determines values *for any* other attribute b, not just some
  - meetid, topic is a superkey for Meetings
  - meetid is a superkey (and a primary key) for Meetings

Meetings

meetid	date	topic	owner
34716	2023-08-28	dDB	ira
34717	2024-01-23	dDB	ira
42835	2023-08-18	TA meeting	aas

People1

owner	name	group	office
ira	Ira Assent	vip	Ny-357
aas	Annika Schmidt	phd	NULL

ira@cs.au.dk

Functional Dependencies

- Which of the following are functional dependencies?
  - topic → date
  - topic → owner
  - owner → name
  - name, owner → topic
  - date → owner

meetid	date	topic	owner
34716	2023-08-28	dDB	ira
34717	2024-01-23	dDB	ira
42835	2023-08-18	TA meeting	aas

owner	name	group	office
ira	Ira Assent	vip	Ny-357
aas	Annika Schmidt	phd	NULL

ira@cs.au.dk

Finding Functional Dependencies

- Given a populated table
  - **Cannot** determine which FDs hold
    - topic → owner?
    - date → topic?
    - topic → date?
  - Requires knowing meaning of and relationships among attributes
  - Can state that FD does *not* hold if there are tuples that show violation of such an FD

meetid	date	topic	owner
34716	2023-08-28	dDB	ira
34717	2024-01-23	dDB	ira
42835	2023-08-18	TA meeting	aas

ira@cs.au.dk

Dependencies Cause Anomalies

Arrangements

meetid	what	date	slot	userid	name	group	office
34716	dDB	2023-08-28	12	ira	Ira Assent	vip	Ny-357
34717	dDB	2024-01-23	14	ira	Ira Assent	vip	Ny-357
42835	TA meeting	2023-08-18	9	aas	Annika Schmidt	phd	NULL

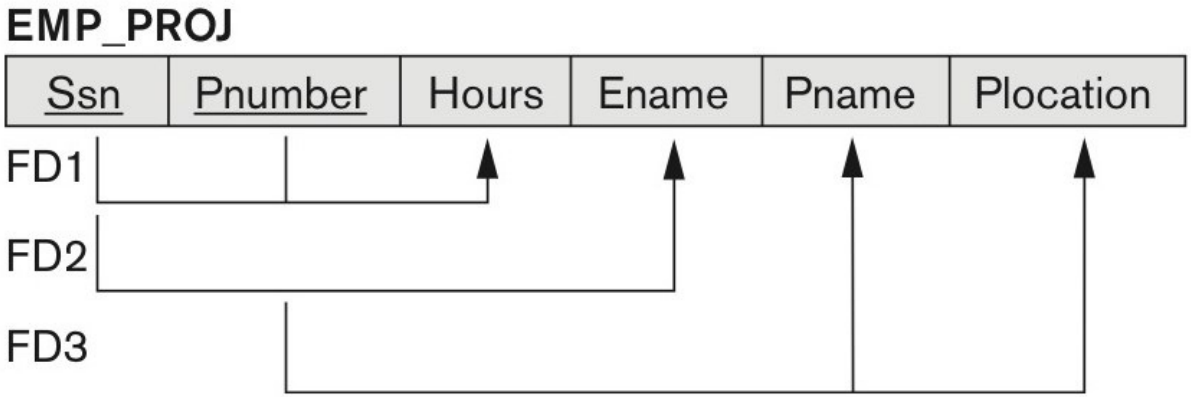
userid $\rightarrow$ name userid $\rightarrow$ group userid $\rightarrow$ office

- If meetid is a key, and userid determines other attributes, there is redundancy
  - every time we enter a userid, we enter values in group, office that can be determined from FD

ira@cs.au.dk

FDs in anomaly-ridden relations

- Mixing employee information with project information



Textbook notation for FDs: line with arrow from left hand side pointing to right hand side of FD

- Ssn, Pnumber → Hours

  (so, if you know who and which project, you know how many hours they spent on the project – they cannot spend two different hour counts on the same project)
- Ssn → Ename
- …

ira@cs.au.dk

Here's the plan

- We want to find all FDs
  - Indicative of potential anomalies
- Use them to decompose "problematic" tables
  - Understand when and how to do that
  - Should retain the same information when used in queries (e.g. no spurious tuples)

ira@cs.au.dk

Properties of FDs

- Some FDs will always hold
- **Trivial**
  - $a_1,...,a_n \rightarrow a_i$ for any set of attributes $a_1,...,a_n$
  - Subset
  - E.g. name, address → name
- **Superkey**
  - $a_1,...,a_n \rightarrow b$ for any b when $a_1,...,a_n$ is a superkey
  - E.g. Meetid, what → date

ira@cs.au.dk

Inference

- Inferring Dependencies
  - Given an initial set of dependencies, find all other ones that follow logically
    - Initially given FDs called **semantically obvious**
    - Logically deduced FDs are called **inferred**

  - For example:
    - If A → B, B→ C, it follows logically that A→ C
      - If fire means heat, heat means explosion, then fire means explosion
      - If userid implies office, office implies department, then userid implies department
    - if A,B → C and A,D → B it follows logically that also: A,D → C

ira@cs.au.dk

Closure of a Set of Dependencies

- **Closure of set of dependencies** *F*
  - denoted as F^+
  - set of all dependencies implied by dependencies in *F*
- Example
  - R = (A, B, C, D)
  - $F = \{A \rightarrow B, A \rightarrow C, CD \rightarrow A\}$
  - Some members of F^+
  - Where BC is short for {B,C} etc.

ira@cs.au.dk

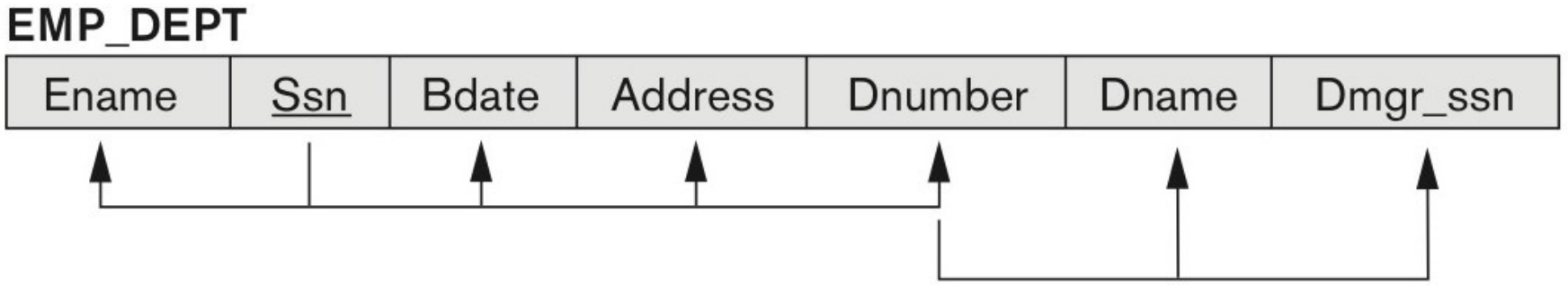
Closure of a Set of Dependencies

- **Closure of set of dependencies** *F*
  - denoted as F^+
  - set of all dependencies implied by dependencies in *F*
- Example
  - R = (A, B, C, D)
  - $F = \{A \rightarrow B, A \rightarrow C, CD \rightarrow A\}$
  - Some members of F^+ $A \rightarrow BC$, $CD \rightarrow B$, $AD \rightarrow B$, $AD \rightarrow ABCD$ …
  - Where BC is short for {B,C} etc.

ira@cs.au.dk

Textbook example inferred FDs

- Given FDs
  - F= { Ssn → { Ename, Bdate, Address, Dnumber },
    Dnumber → { Dname, Dmgr_ssn } }



- Examples of inferred FDs
  - F^+ = {Ssn → {Dname, Dmgr_ssn}, Ssn → Ssn, Dnumber → Dname, …}

ira@cs.au.dk

What is NOT true?

1. $C \rightarrow D$ and $D \rightarrow E$, then $C \rightarrow E$
2. $DC \rightarrow D$
3. $DC \rightarrow B$, *then* $D \rightarrow B$ *or* $C \rightarrow B$
4. $C \rightarrow D$, then $BC \rightarrow BD$

ira@cs.au.dk

Armstrong's axioms / rules

- Armstrong's rules help compute closures
  1. *Reflexivity*: If $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ (for any sets of attributes α and β)
  2. *Augmentation*: If $\alpha \rightarrow \beta$ then $\gamma\alpha \rightarrow \gamma\beta$
  3. *Transitivity*: If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$

- Examples

ira@cs.au.dk

Armstrong's axioms / rules

- Armstrong's rules help compute closures
  1. *Reflexivity*: If $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ (for any sets of attributes α and β)
  2. *Augmentation*: If $\alpha \rightarrow \beta$ then $\gamma\alpha \rightarrow \gamma\beta$
  3. *Transitivity*: If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$

- Examples
  1. $DC \rightarrow D$
  2. $C \rightarrow D$, then $BC \rightarrow BD$
  3. $C \rightarrow D$ and $D \rightarrow E$, then $C \rightarrow E$

ira@cs.au.dk

Computing Closure

- Algorithm

  Initially $F^+ = F$

  Repeat until no more FDs can be added to F^+

  apply an Armstrong rule to FDs in F^+

  add result to F^+

- Example
  - Given the following functional dependencies F, compute F^+
    - $F = \{ A \rightarrow BC, \quad CD \rightarrow E, \quad B \rightarrow D, \quad E \rightarrow A \}$

  $E \rightarrow A$ and $A \rightarrow BC$, so $E \rightarrow BC$ (transitivity)

  $B \rightarrow D$, so $CB \rightarrow CD$ (augmentation)

  $CB \rightarrow CD$ and $CD \rightarrow E$, so $CB \rightarrow E$ (transitivity)

  ...

ira@cs.au.dk

Sound and complete

- Armstrong's Axioms are
  - Sound: generate only correct functional dependencies
  - Complete: generate all possible FDs (F^+) from F
    - So, if you keep applying them to F, you will end up with all of F^+
- Additional rules
  - Follow from the above, so not necessary, but may be convenient to use:
  4. *Union*: If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ then $\alpha \rightarrow \beta\gamma$
  5. *Decomposition*: If $\alpha \rightarrow \beta\gamma$ then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$
  6. *Pseudotransitivity*: If $\alpha \rightarrow \beta$ and $\gamma\beta \rightarrow \delta$ then $\alpha\gamma \rightarrow \delta$
- Examples
  4. C → D and C → B then C → BD
  5. C → BD then C → B and C → D
  6. C → D and AD → B then CA → B

ira@cs.au.dk

Practical approach

- Derive functional dependencies from miniworld
- Find each set of attributes on left hand side
  - Find all attributes dependent on them
- Again, we find a closure, this time on attributes

- Algorithm:
  - Given FDs F on relation R with attributes A, subset X of A
  - Initially $X^+ = X$
  - For each FD Y → Z in F do
    - if $X^+ \supseteq Y$ then $X^+ := X^+ \cup Z$;
  - Until X^+ unchanged

ira@cs.au.dk

Attribute Closure example

We are given relation R, and the set of functional dependencies F from the miniworld

$R = (A, B, C, D), \quad F = \{A \rightarrow B, A \rightarrow C, CD \rightarrow A\}$

Obtain the attribute closure for attributes A and D as follows:

AD^+: $result = AD$ (initial set of attributes given)

$result = ADB$ ($A \rightarrow B$ and $A \subseteq AD$)

$result = ADBC$ ($A \rightarrow C$ and $A \subseteq ABD$)

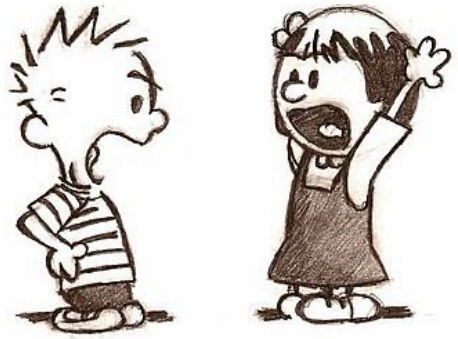
ira@cs.au.dk

Closure and keys

$R = (A, B, C, D), \quad F = \{A \rightarrow B, A \rightarrow C, CD \rightarrow A\}$

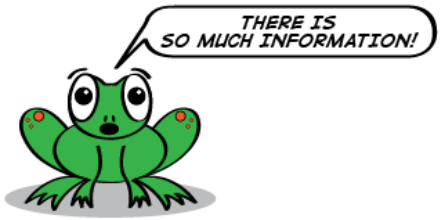
$AD^+ = ADBC$

1. AD is a superkey and a candidate key
2. AD is a candidate key, but not a superkey
3. AD is a superkey, but not a candidate key
4. AD is neither a superkey nor a candidate key



ira@cs.au.dk

Textbook example FDs



CLASS (Classid, Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity)

- FDs: FD1: Classid → Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity
  - Semantics: unique classid, is a key for all other attributes
- FD2: Course# → Credit_hrs;
  - Semantics: each course only has a single number of credits (regardless of e.g. different study programs etc that the course may be part of)
- FD3: {Course#, Instr_name} → Text, Classroom;
  - Semantics: When we know the course and the teacher, we will know which textbook and which room are used (so different teachers may use different books for the same course, but not different books for the same teacher for a given course)
- FD4: Text → Publisher
  - Semantics: When we know a textbook, we also know who is the publisher of that textbook (so the textbook cannot move from one publisher to another)
- FD5: Classroom → Capacity
  - Semantics: A classroom serves a fixed number of people

ira@cs.au.dk

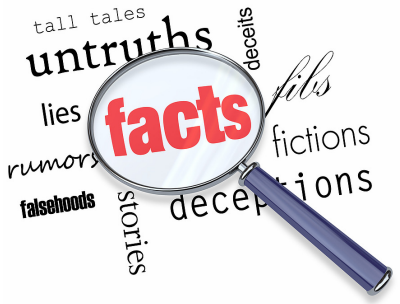
Textbook example on derived FDs

- Attribute Closure not only useful in determining FDs
  - Can also help understand semantics (meaning) of attributes and how they relate further
- {Course#}+ = {Course#, Credit_hrs}
  - So the course# only determines the number of credits, but not the instructor or any other information, which would make the classes differ
- {Course#,Instr_name}+ = {Course#, Credit_hrs, Text, Publisher, Classroom, Capacity}
  - Classid not included, so
  - Course#, Instr_name is not a key:
  - a course# with different instructors would make them distinct classes

ira@cs.au.dk

Summary

- Intended learning outcomes
- Be able to
  - Identify anomalies and how they relate to functional dependencies
  - Work with Functional Dependencies



ira@cs.au.dk

Where to go from here?

- We know that anomalies are caused by dependencies
- We know how to infer functional dependencies

- Now let's use this to improve the schema
  - Normal forms

ira@cs.au.dk

What was this all about?

Guidelines for your own review of today's session

- Poor design can lead to the following anomalies…
  - To counter these anomalies, we have guidelines that say…
- Dependencies describe…
  - We determine them in order to …
- Functional dependencies are related to keys in that…
  - They are defined as follows…
  - A functional dependency is found by…
  - The closure is…
  - It is computed by…

ira@cs.au.dk