

Archiving Scientific Data

Michael Haspra



Introduction

- Paper:
“Archiving Scientific Data”, P. Buneman et al., 2002
- Topic:
An archiving technique for hierarchical data with key structure.
- More precise:
An archiving tool for XML data.

Problem statement

- Scientific research makes extensive use of databases which go through different states during the research process.
 - For example in drug development.
- These states of the data have to be archived!
- Lack of data archiving may leads to:
 - Non-repeatable and therefore not verifiable findings.
 - Undetected results.

What already existed

- Diff based techniques:
 - Store a base version together with edit scripts.
 - Typically documents and descriptions about line changes.
- But they have some problems.

Problems of diff based techniques

- Performance problem
 - Processing many edit scripts is costly.
- Semantic problem
 - No “object” continuity, because there are no keys.

Name: Ann
Address: Annstreet
Name: Bob
Address: Bobway

What has changed?

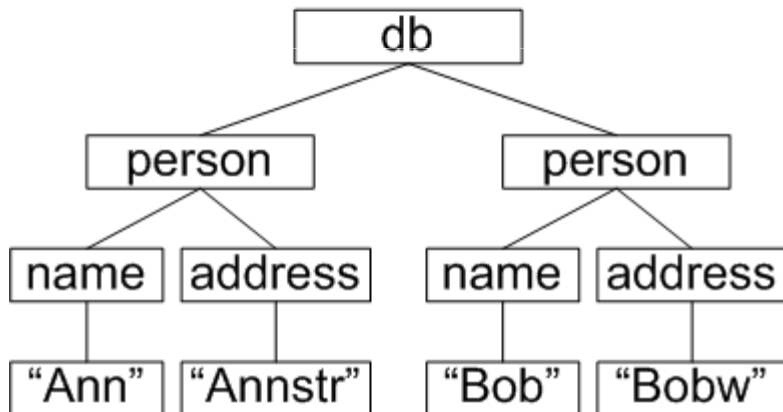
Name: Bob
Address: Annstreet
Name: Ann
Address: Bobway

The proposed archiving technique

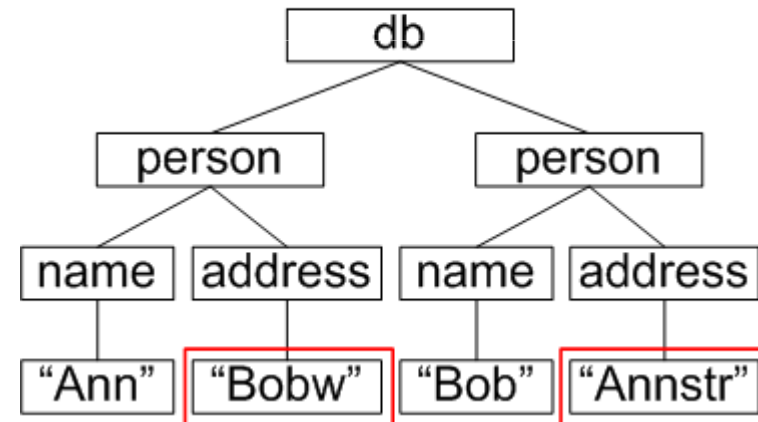
- Gives a solution to three questions:
 - How to store the data?
 - Store all versions in a big tree.
 - How to save changes of data?
 - Annotate keys + nested merge.
 - How to retrieve data?
 - Fast retrieval with timestamp trees.

How to store the data?

Version 1

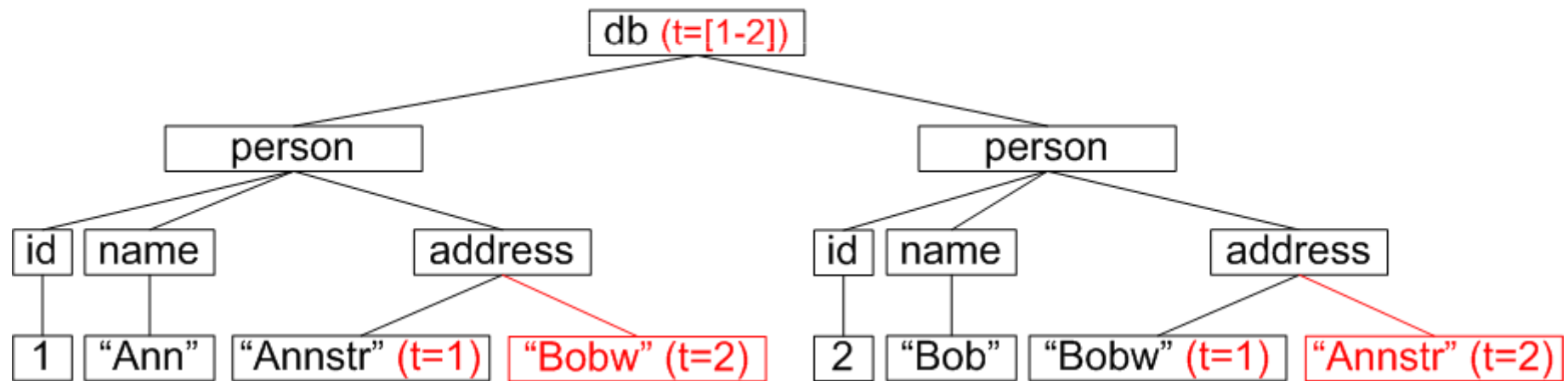


Version 2



How to store the data?

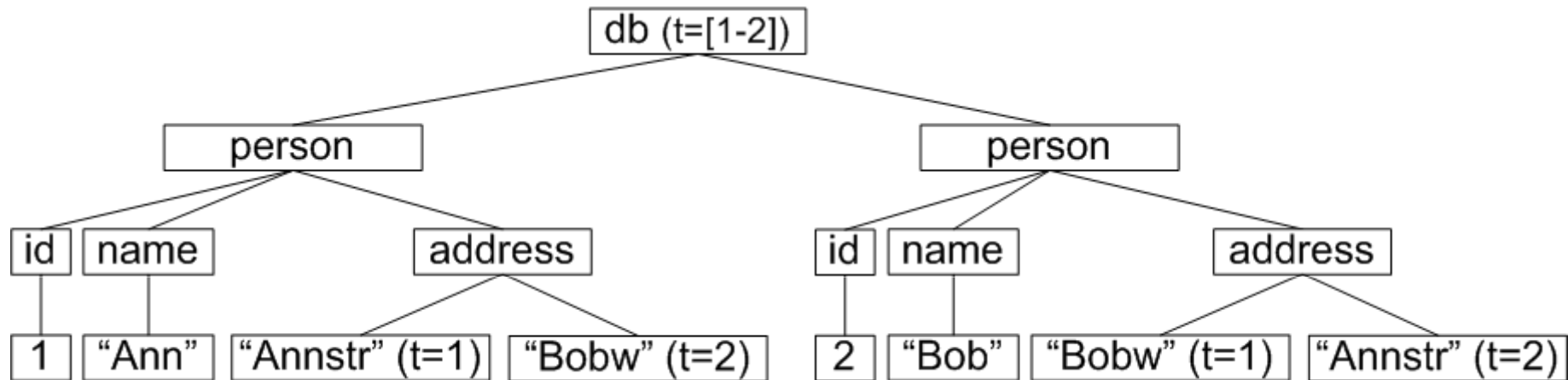
Store all versions in a big tree.



Keys

- A path is a sequence of node names
 - “/db/person/name”
- A key is
 - A pair $(Q, \{P_1, \dots, P_n\})$
 - Where Q is the so called context-path.
 - And $\{P_1, \dots, P_n\}$ are the so called target paths.
 - Every node defined by $\{P_1, \dots, P_n\}$ has to be present and unique in Q .

Keys in the example



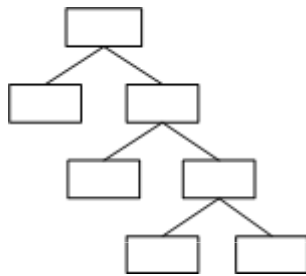
$(/, (db, \{\}))$ There is one db element at the root.

$(/ db / person, \{id\})$ Every person can be uniquely identified by its id.

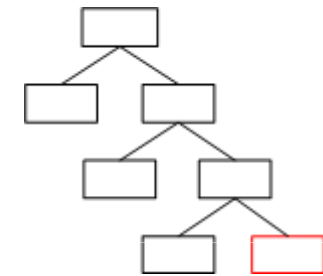
Which nodes are keyed is application specific!

How to save changes of data?

Existing archive



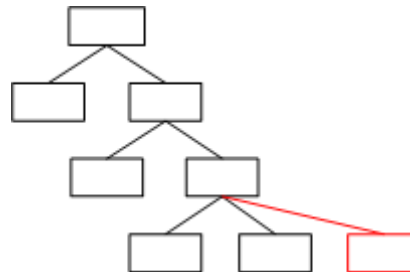
New version



Annotate
Keys

Nested
Merge

Resulting archive



Nested Merge

- The (simplified) algorithm:

Nested_Merge(x, y, T)

if timestamp for x exists then extend it by i

Let X and Y denote the set of all children
nodes of x and y respectively

if y is a frontier node then

merge the sub tress

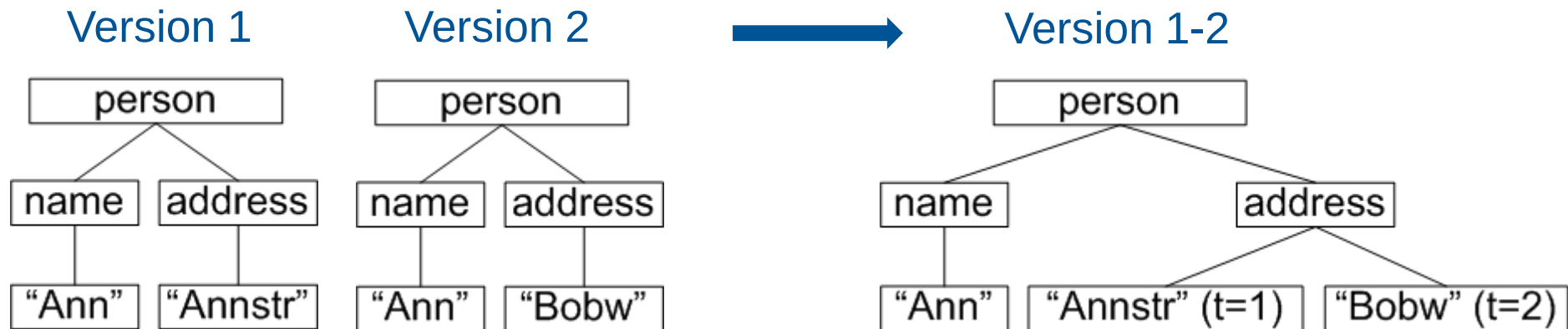
else

divide X and Y in 3 sets XY , X' and Y'

process XY , X' and Y'

if y is a frontier node

```
if y is a frontier node then
  if every node in X is not timestamped then
    if content of x and y is different then
      add content of y to x and update timestamps
  else
    update nodes which stayed the same and add
    the new nodes
```



if y is not a frontier node

else

Let $XY = \{(x', y') \mid x' \notin X, y' \notin Y, x' \text{ and } y' \text{ have same label and key}\}$

Let $X' = \{x \in X \mid (\forall y \in Y, (x, y) \notin XY)\}$

Let $Y' = \{y \in Y \mid (\forall x \in X, (x, y) \notin XY)\}$

for every pair in XY

 apply nested merge

for every x' in X'

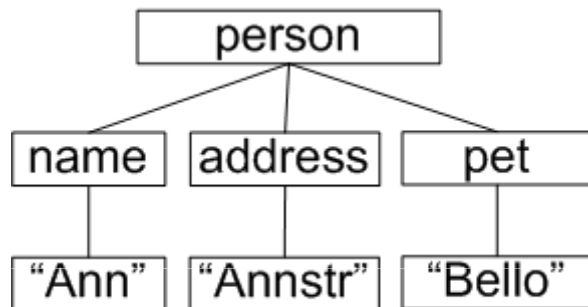
 if x' has no timestamp add T- $\{i\}$

for every y' in Y'

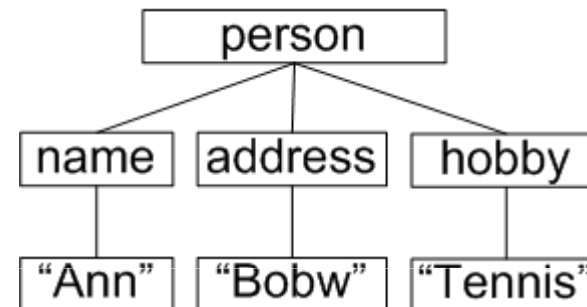
 add a timestamp i and add y' as child of x

if y is not a frontier node

Version 1



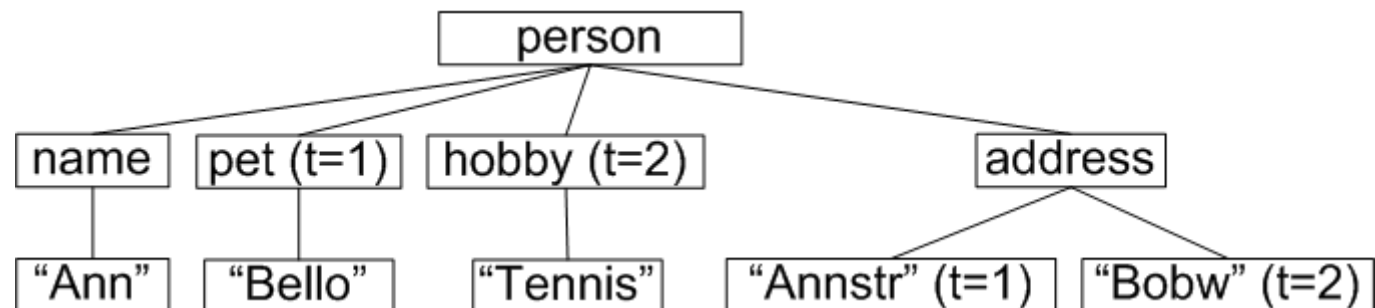
Version 2



$XY = \{\text{name}, \text{address}\}$

$X' = \{\text{pet}\}$

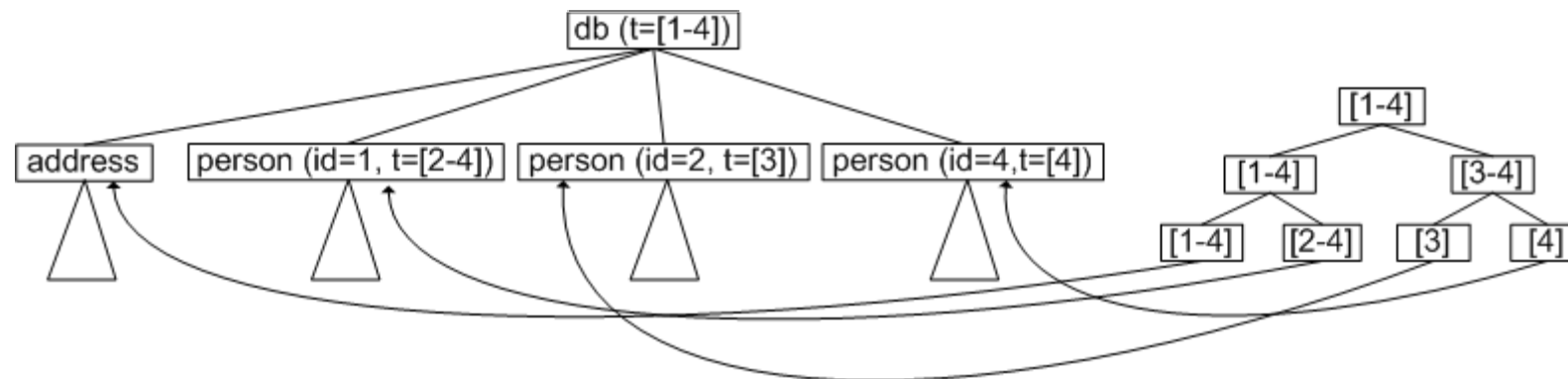
$Y' = \{\text{hobby}\}$



How to retrieve data?

- One could just scan and look for timestamp.
 - This would be too slow.
- Idea: Auxiliary timestamp data structures
 - The timestamp trees.
 - For **every** keyed node a binary tree containing the timestamps of its child nodes is constructed.
 - These trees are reconstructed **every time** nested merge is applied.

Timestamp tree



Retrieving

- Retrieving is now done by:
 1. Get location of needed timestamp trees.
 2. Get all valid nodes by traversing the timestamp trees.
 3. Get the nodes from the archive.

Some criticism

- Requirements to the data
 - Not all data support such a key system.
 - Not all data is hierarchical.
- Storage
 - May need lots of space because of the auxiliary data structures.
- Nested Merge
 - Always applied to the whole archive.
 - The timestamp trees are recomputed every time.
 - May processes the same node several times.
 - What if the data does not fit in main memory?

XArch

- An application to archive XML documents.
- Published 6 years later.
- Solves some issues with nested merge but needs more auxiliary data structures.
 - Idea: Store a sorted list of children for every keyed node.
- Has a query language.
- XArch is still being developed (<http://xarch.sourceforge.net>)