Managing Historical Retention in Database Systems

Gerome Miklau

Joint work with Brian Levine, Patrick Stahlberg, Wentian Lu
History has benefits

History: a stored record of data and operations performed on a system.

• Arguments for preserving history
  • Protection against loss
  • History is useful: accountability
  • Storage is cheap

holding people/programs responsible for actions taken.
History has risks

• Arguments **against** preserving history
  
  • Persistence threatens privacy.
  
  • Institutions can be compelled to reveal retained data (even if they don’t want to).
  
  • There are significant benefits to **institutional forgetfulness**.
## Retention policies

**Privacy** and **accountability** balanced through retention policies

<table>
<thead>
<tr>
<th>Institution</th>
<th>Collected Info</th>
<th>Retention Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Russian KGB</td>
<td>speech, actions, etc.</td>
<td>&quot;хранить вечно&quot; (&quot;to be preserved forever&quot;)</td>
</tr>
<tr>
<td>Credit agency</td>
<td>late payments, defaults, etc.</td>
<td>7 years</td>
</tr>
<tr>
<td>Google</td>
<td>search engine queries</td>
<td>18 months</td>
</tr>
</tbody>
</table>

[Mayer-Schoenberger 2007]
Securing history

Central issue: how and when historical data is retained in systems, who can recover and analyze it.

- To support **privacy**: “memory-less” systems
- To support **accountability**: preserve needed history efficiently, permit analysis, protection mechanisms.
Databases don’t forget

Unintentionally retained data is recovered by forensic analysis.

A forensic investigator is a powerful adversary:

- access to persistent storage at time \( t \)
- goal: recover expired data and/or history of operations

*(Threats to Privacy in the Forensic Analysis of Database Systems. SIGMOD 2007)*
Propagation of sensitive data

- **INSERT** sensitive record
- (later) **DELETE** the record
  - deletion is “logical” -- data is not destroyed
- actual persistence of data is hard to predict, and virtually impossible to control.
Slack data in table storage

Delete t3, t5

Insert t7

Delete t1, t4

Vacuum

Deletion is insecure
Vacuum is insecure

Database slack
Filesystem slack
Experiments

• We studied:

- Built forensic recovery tools which scan database pages, recovering expired tuples.

• Table storage
  • deletion is insecure in all systems
  • database and file system slack data generated in proportion to
    • workload, vacuum, clustering.
Recoverable database slack

![Graph showing the relationship between operations and records in Slack]

- Expired records
- MySQL (MyISAM)
- DB2
- SQLite
- MySQL (InnoDB)
- PostgreSQL
Recoverable database slack

![Graph showing recoverable database slack over operations for different database engines.]
Recoverable database slack

![Graph showing recoverable database slack over operations]

- **# records in Slack (x1000)**
- **operations (x1000)**

The graph compares the number of records in Slack over operations for different database systems:
- **Expired records**
  - MySQL (MyISAM)
  - MySQL (InnoDB)
  - DB2
  - SQLite
  - PostgreSQL

The black line represents MySQL (MyISAM), the purple line represents MySQL (InnoDB), the blue line represents DB2, and the red line represents SQLite.
Recoverable database slack

![Graph showing recoverable database slack](image)

- **# records in Slack (x1000)**
- **operations (x1000)**

Legend:
- Expired records
- MySQL (MyISAM)
- DB2
- MySQL (InnoDB)
- PostgreSQL
- SQLite
Recoverable database slack

![Graph showing the number of records in Slack over operations for different databases]

- **MySQL (MyISAM)**
- **MySQL (InnoDB)**
- **DB2**
- **SQLite**
- **PostgreSQL**
Recoverable database slack

The graph depicts the number of records in Slack (multiplied by 1000) against operations (also multiplied by 1000) for different database management systems. The systems compared are:

- Expired records
- MySQL (MyISAM)
- SQLite
- MySQL (InnoDB)
- DB2
- PostgreSQL

The graph shows a linear increase in records for each database system with an increase in operations.
Other system components

• **Indexes**
  
  • Sequence of past operations that led to current state may be revealed by:
    
    • structure, physical representation (in memory or on disk)
  
  • B+Trees are not history-independent

• **Transaction log**
  
  • Log usually contains the before and after image of each DB modification
  
  • Bounds on retention depend on:
    
    • workload, checkpointing frequency, size of log device, etc.
Problem with forensic data recovery

• Intended interface of database (SQL) does not reliably represent the stored contents of the database
  • e.g. deleted tuples do not appear in query results, but are recoverable.
  • tuples do not have “age” or order in data model, but this info can be recovered from disk image.
Transparent systems

Clarity of interfaces

• The system should provide users with clear, accurate bounds on the persistence of data in the system.

Purposeful retention

• Data retained after deletion must have a legitimate purpose, and data should be removed once that purpose is no longer valid.

Complete removal

• Deleted data must be destroyed, including copies and derived versions.
Secure deletion in DBMS

- Two basic strategies for secure deletion:
  - overwrite data with zeroes
  - store data in encrypted form, delete by disposing of keys.

- For table storage:
  - pages are read and written often
  - prefer secure deletion and vacuum using overwriting

- For transaction log:
  - sequential writes, easily identifiable point of expiry
  - use encryption with key disposal
Databases can remember, but not safely

- **Existing capabilities**
  - Transaction logs, audit logs, point-in-time recovery
  - Postgres, temporal DBs, transaction-time DBs

- **Limitations**
  - Insufficient information retained, inefficient access
  - All-or-nothing protection model
Audit queries

• Audit the history of modifications to the database
• Note: we are not auditing database reads.
• For example:
  • What was Bob’s lowest salary?
  • How many times was Bob’s salary changed?
  • Who made the last update to Bob’s salary?
A transaction-time data model

### Audit Log

<table>
<thead>
<tr>
<th>User</th>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Insert</td>
<td>1995</td>
</tr>
<tr>
<td></td>
<td>(Bob,50k)</td>
<td></td>
</tr>
<tr>
<td>Joe</td>
<td>Update</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>Bob salary=60k</td>
<td></td>
</tr>
</tbody>
</table>

### Database

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>50k</td>
<td>1995</td>
<td>2000</td>
</tr>
<tr>
<td>Bob</td>
<td>60k</td>
<td>2000</td>
<td>2008</td>
</tr>
</tbody>
</table>

- What was Bob’s lowest salary? 50k
- How many times was Bob’s salary changed? 1
- Who made the last update to Bob’s salary? Joe
Retention policy

- Policies limiting retention require removing parts of history.
  - Expunge particular records, time periods, etc.
  - Redact records (by removing sensitive values)
  - Compress time periods by summarization

**Example Policy:**
Redact Bob’s salary prior to 2002

- Intuition: we shouldn’t need to know the value of Bob’s salary to perform interesting audit queries.
Transforming history

<table>
<thead>
<tr>
<th>User</th>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Insert</td>
<td>1995</td>
</tr>
<tr>
<td>Joe</td>
<td>Update</td>
<td>2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>50k</td>
<td>1995</td>
<td>2000</td>
</tr>
<tr>
<td>Bob</td>
<td>60k</td>
<td>2000</td>
<td>2008</td>
</tr>
</tbody>
</table>

What was Bob’s lowest salary? unknown

How many times was Bob’s salary changed? 1

Who made the last update to Bob’s salary? Joe
Challenges

• A representation system for an **incomplete** audit log

• Answering audit queries over incomplete history

• Deeper issue: how much information can we preserve for accountability, while achieving the privacy goals of the retention policy?

• Note that temporal incompleteness also occurs when:
  • Auditor has imperfect observations of the past.
  • Efficiency concerns mean we can’t store everything.
Conclusion

- History should be a “first-class” part of a DBMS
- The safe, accurate configuration of the system’s historical memory allows needed balance between privacy and accountability.

Transparency requirements:
- Interface should faithfully represent stored contents.

Auditing and retention:
- Techniques to sanitize history while preserving auditing capabilities.
Questions?