

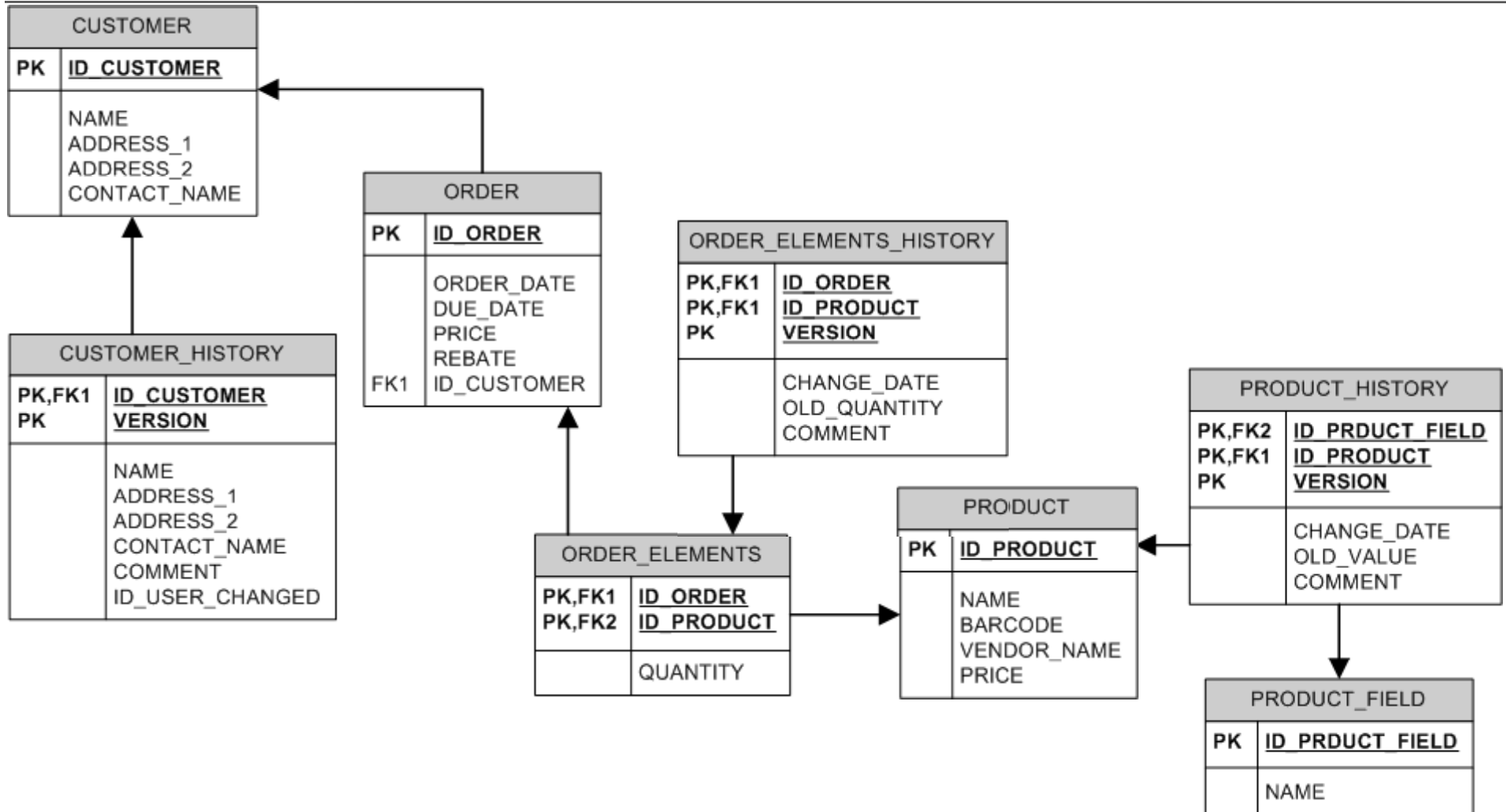
Automated data versioning

Peter Benjamin Volk^{* +}, Guillaume Delannoy^{*}

^{*}DDEngine.org

⁺Technische Universität Dresden
Database Technology Group

A typical OLTP application



Requirements

*Generic method for **all** DML statements on Table*

Versioning with low overhead for DML

Schema optimal for workload and storage requirements

*Versioned data accessible via **SELECT** statements*

Low maintenance

No “own” storage engine

Stakeholders

- Required for data security
- Legal regulations
- Internal provisioning

Real world example

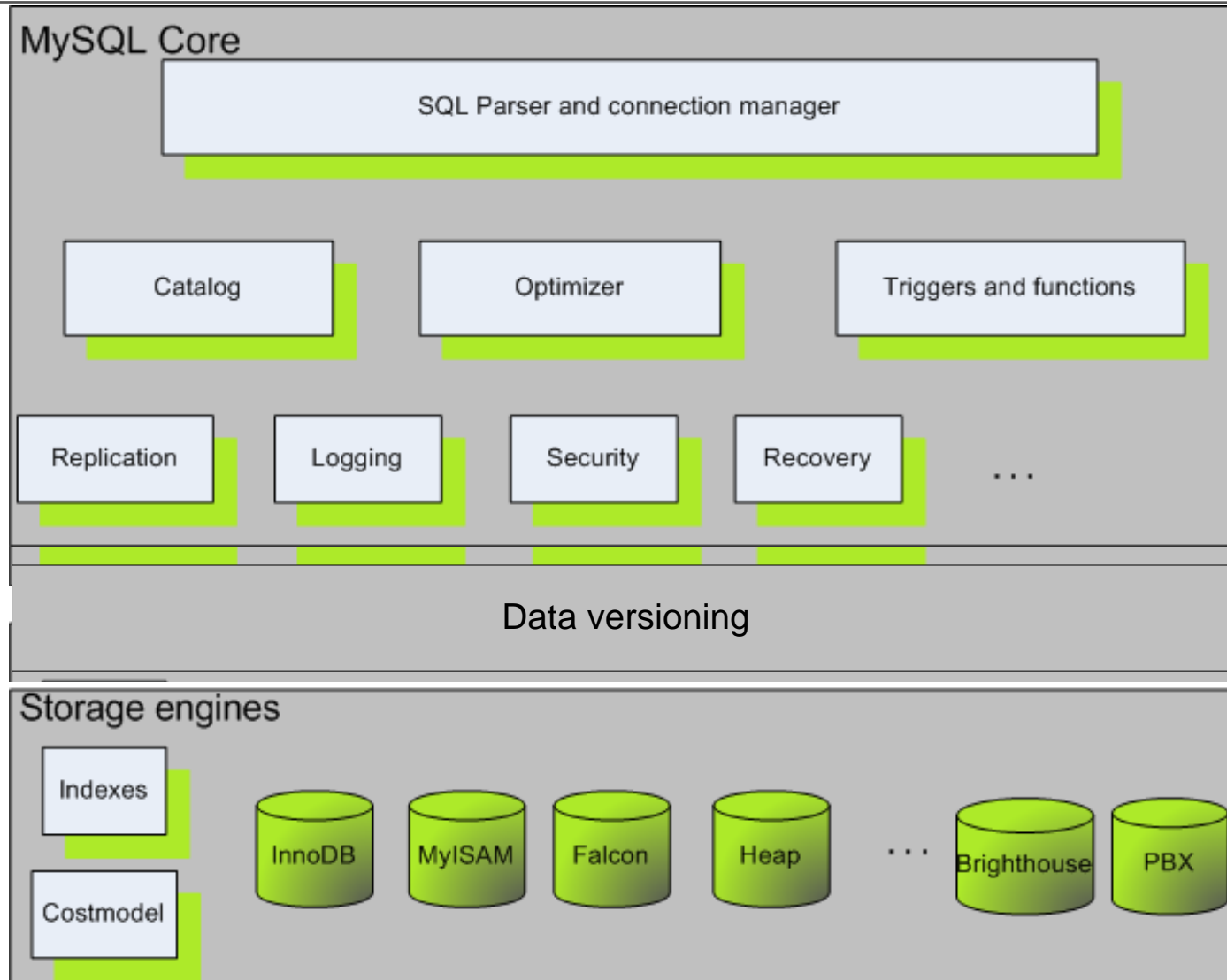
- 1200 Tables
- 43% tables for versioning tables
- Code for versioning most complicated in application and least maintained
- Low TX volume on versioning tables

Multiple table versioning schemas

Implementation via

- Triggers on tables
 - High maintenance
- Logic in application layer
 - Multiple applications on same DB
 - Complicated code
- Query log extraction
 - High storage effort
 - Roll forward only
- WL#2878 (Simple data auditing)
 - MySQL 7.0 (maybe)

Architecture

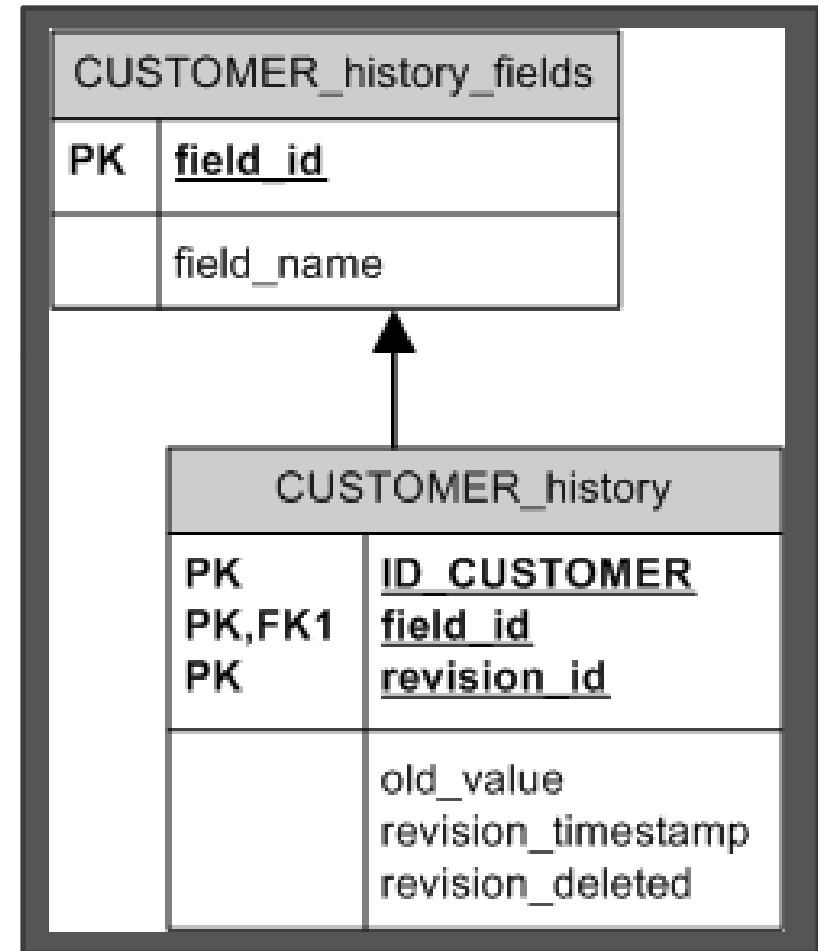


Versioning schemas (I)

CUSTOMER_history	
PK PK	<u>ID CUSTOMER</u> <u>revision id</u>
	FNAME SNAME STREET CITY ZIP CREATOR revision_timestamp revision_deleted

SINGLE

CUSTOMER	
PK	<u>ID CUSTOMER</u>
	FNAME SNAME STREET CITY ZIP CREATOR

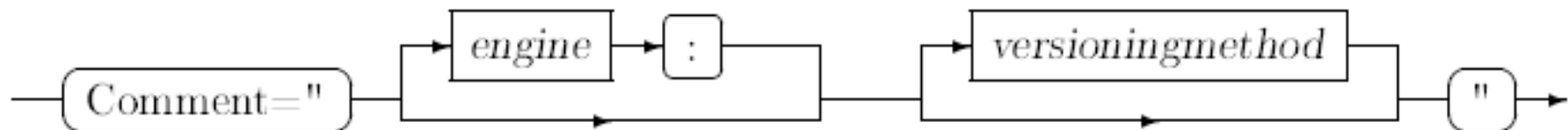


DOUBLE

Table creation

Revision engine as “normal” storage engine

```
create table CUSTOMER (  
  ID_CUSTOMER int not null primary key,  
  FNAME char(20),  
  SNAME char(20),  
  STREET char(50),  
  CITY char(50),  
  ZIP int(10),  
  CREATOR char(50)  
) engine=revision comment="InnoDB:SINGLE";
```



Insert, Update, Delete

Insert

- Insertion into base table

Update

- *Marking of change properties in data set*
- *Insertion old values into history table*
- Update on base table

Delete

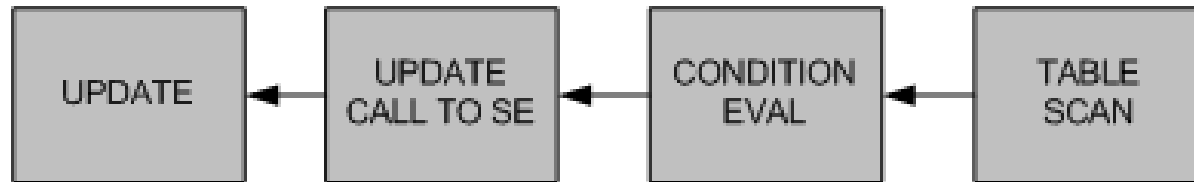
- Insertion of row into history table
- Mark row as deleted in history table
- Delete on base table

Versioning

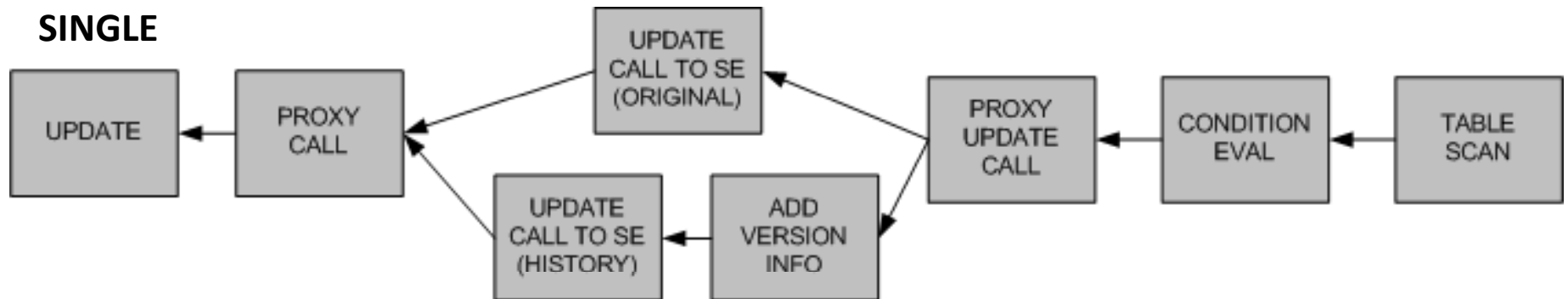
- Version ID is autoincrement per row
- No DML statements at all on versioning tables

Query expansion (Update)

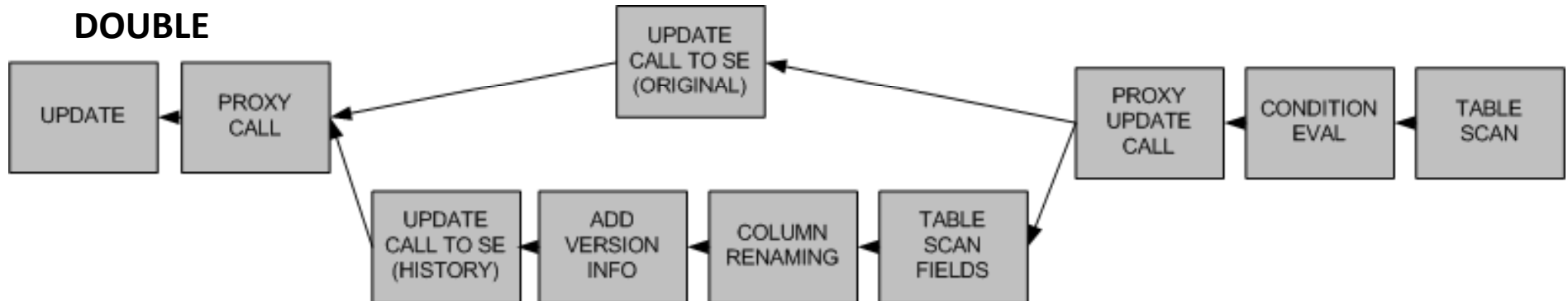
Original



SINGLE

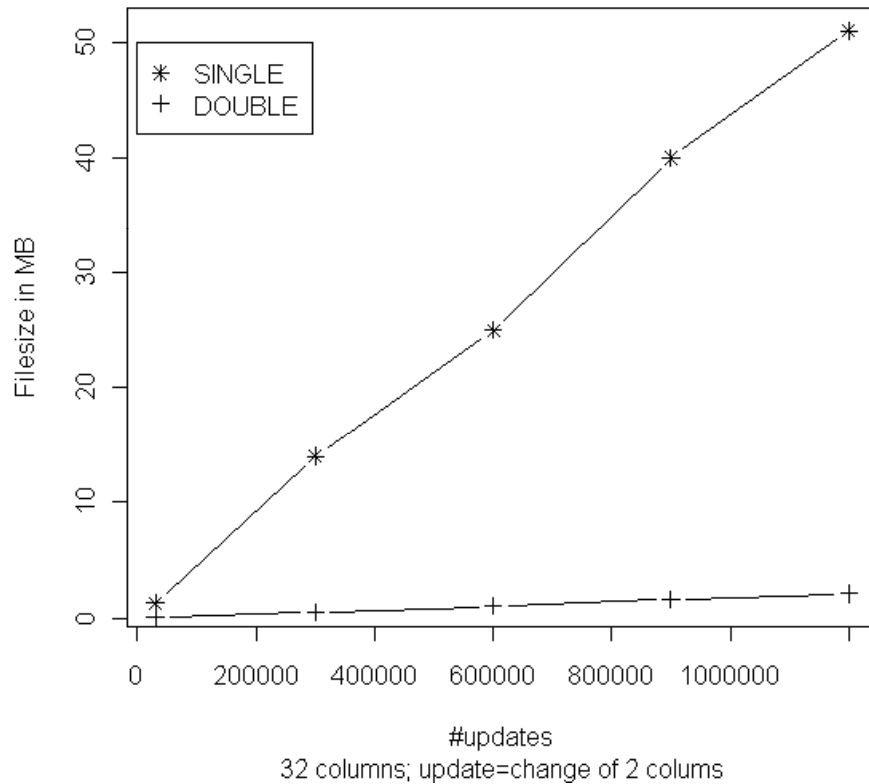


DOUBLE

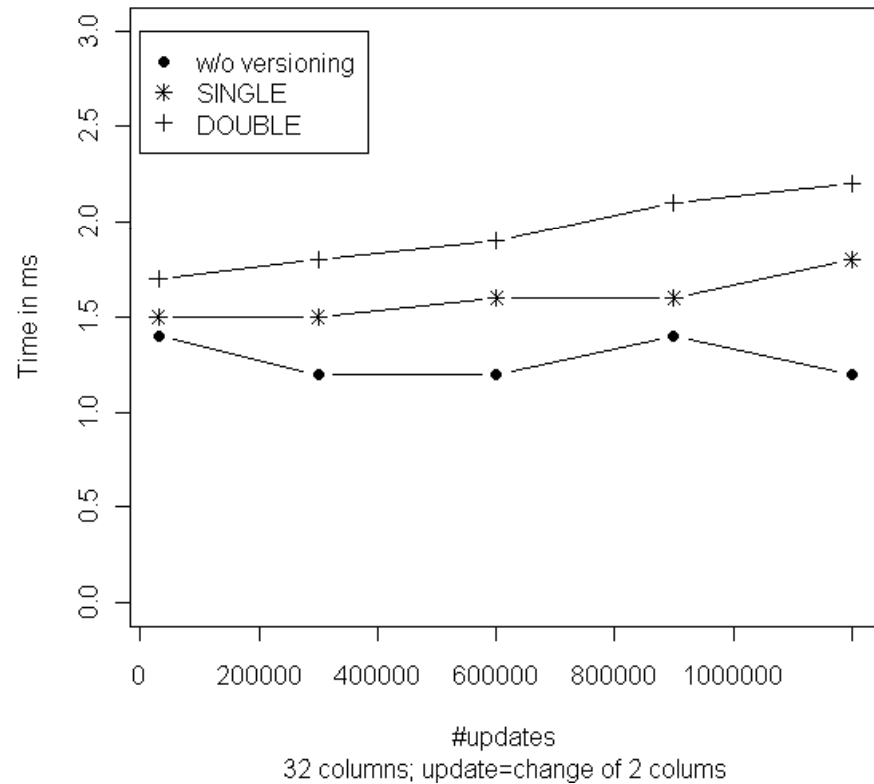


Evaluation (I)

File size vs. number of update statements



Update speed vs. number of update statements



*Data Version selectable via session variable
(**revision_select_mode**)*

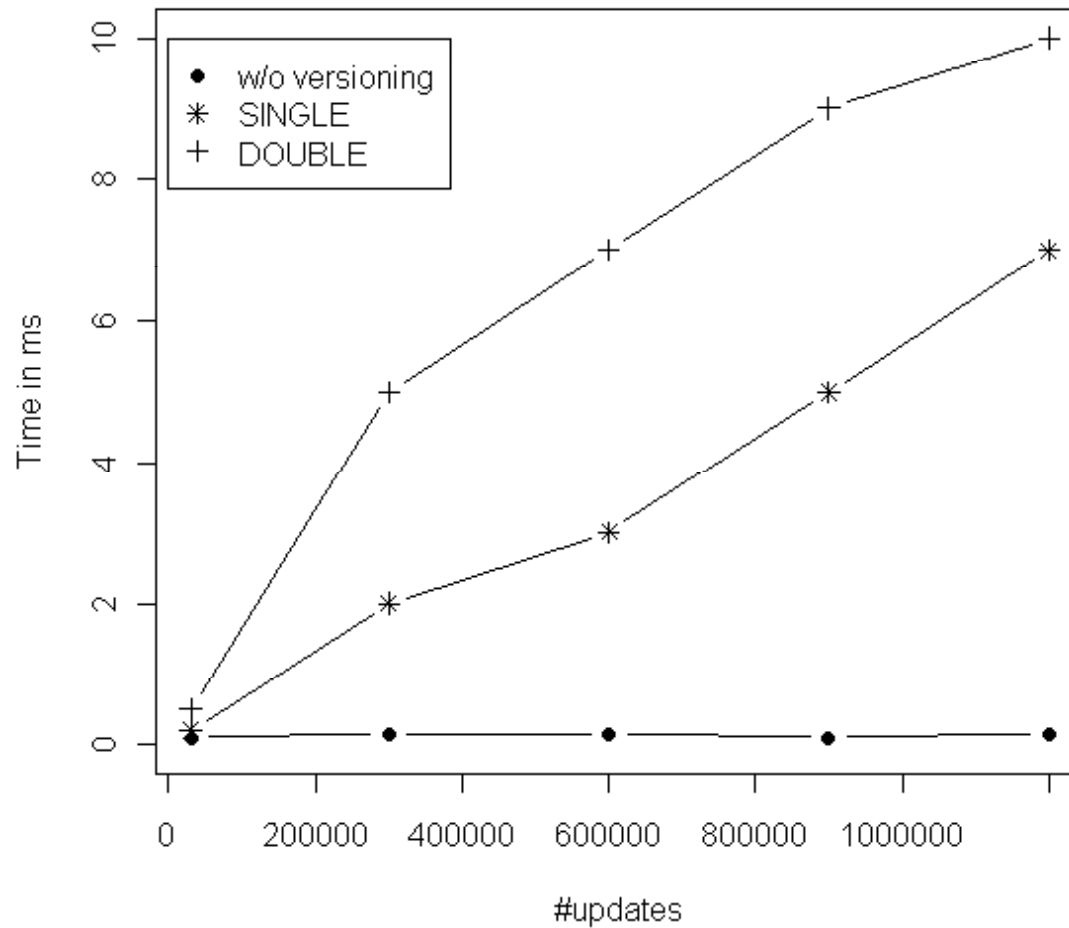
- current (default)
- history → All historic data
- *deleted* → All deleted data sets
- *time* → Data set to a specific time

*Extended Selection (**revision_select_asof**,
revision_select_mode)*

- time → specifies the exact point of time of the version

Evaluation (II)

Select speed vs. number of update statements



32 columns; update=change of 2 columns

Versioning schemas (II)

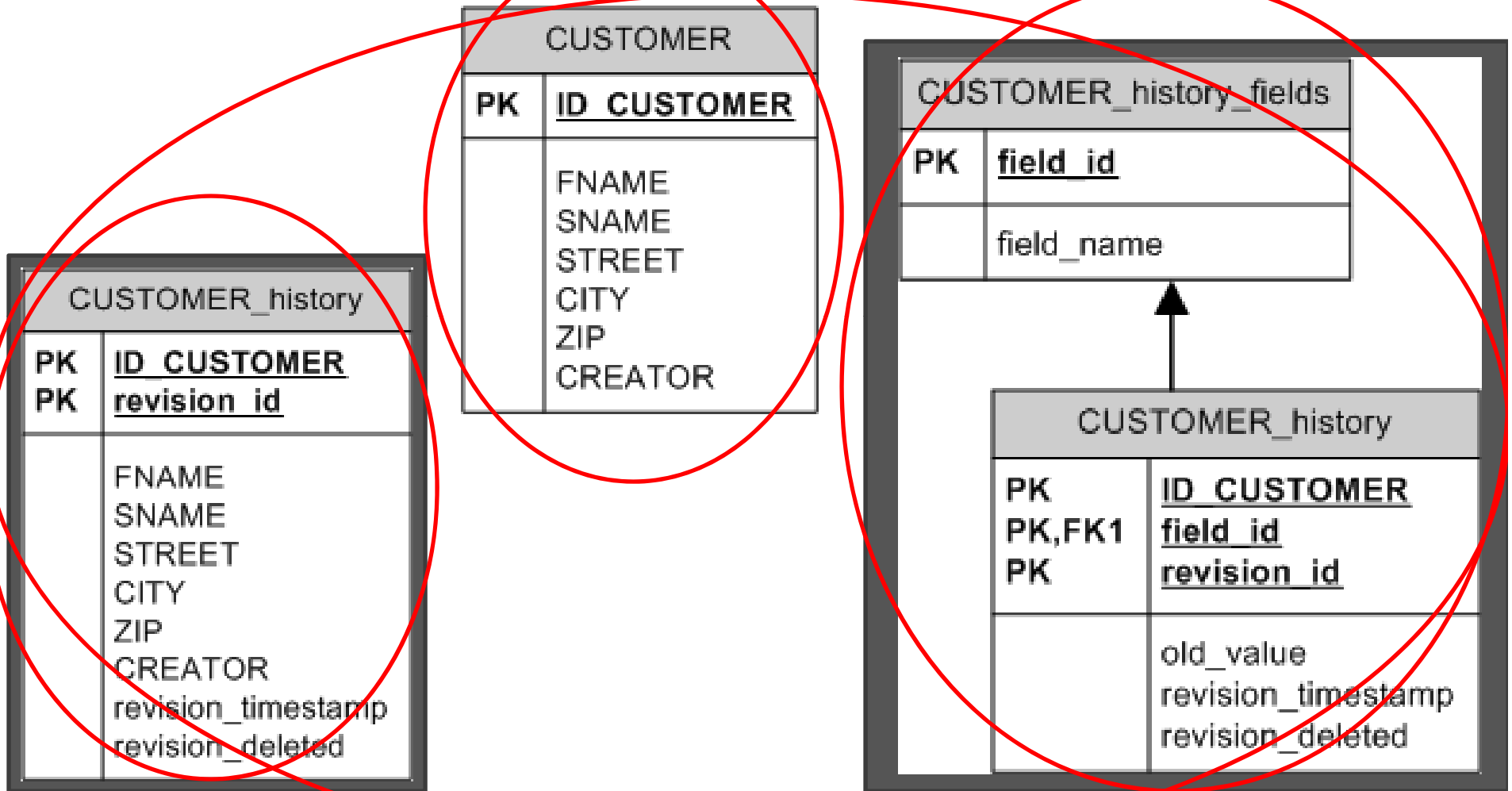
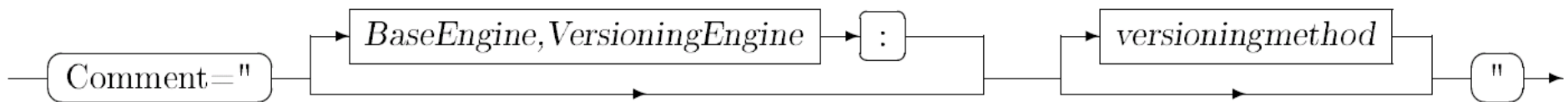


Table creation (II)

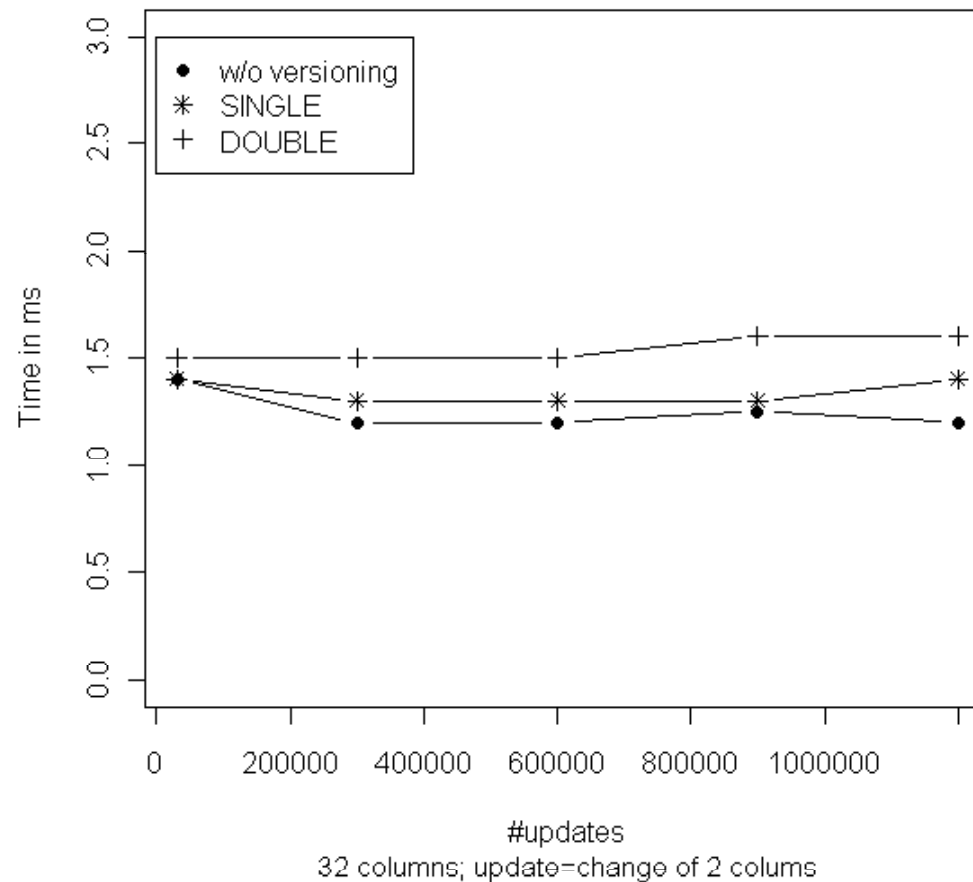
Revision engine as “normal” storage engine

```
create table CUSTOMER (  
  ID_CUSTOMER int not null primary key,  
  FNAME char(20),  
  SNAME char(20),  
  STREET char(50),  
  CITY char(50),  
  ZIP int(10),  
  CREATOR char(50)  
  ) engine=revision comment="InnoDB,Archive:DOUBLE";
```



Evaluation (III)

Update speed vs. number of update statements



Automatic table type selection

```
engine=revision comment="InnoDB,AUTO:DOUBLE";
```

Versioning tables rarely selected

Tradeoff between select/insert performance and disk consumption

If Versioning tables selected then for long time period → automated table transformation

Tradeoff

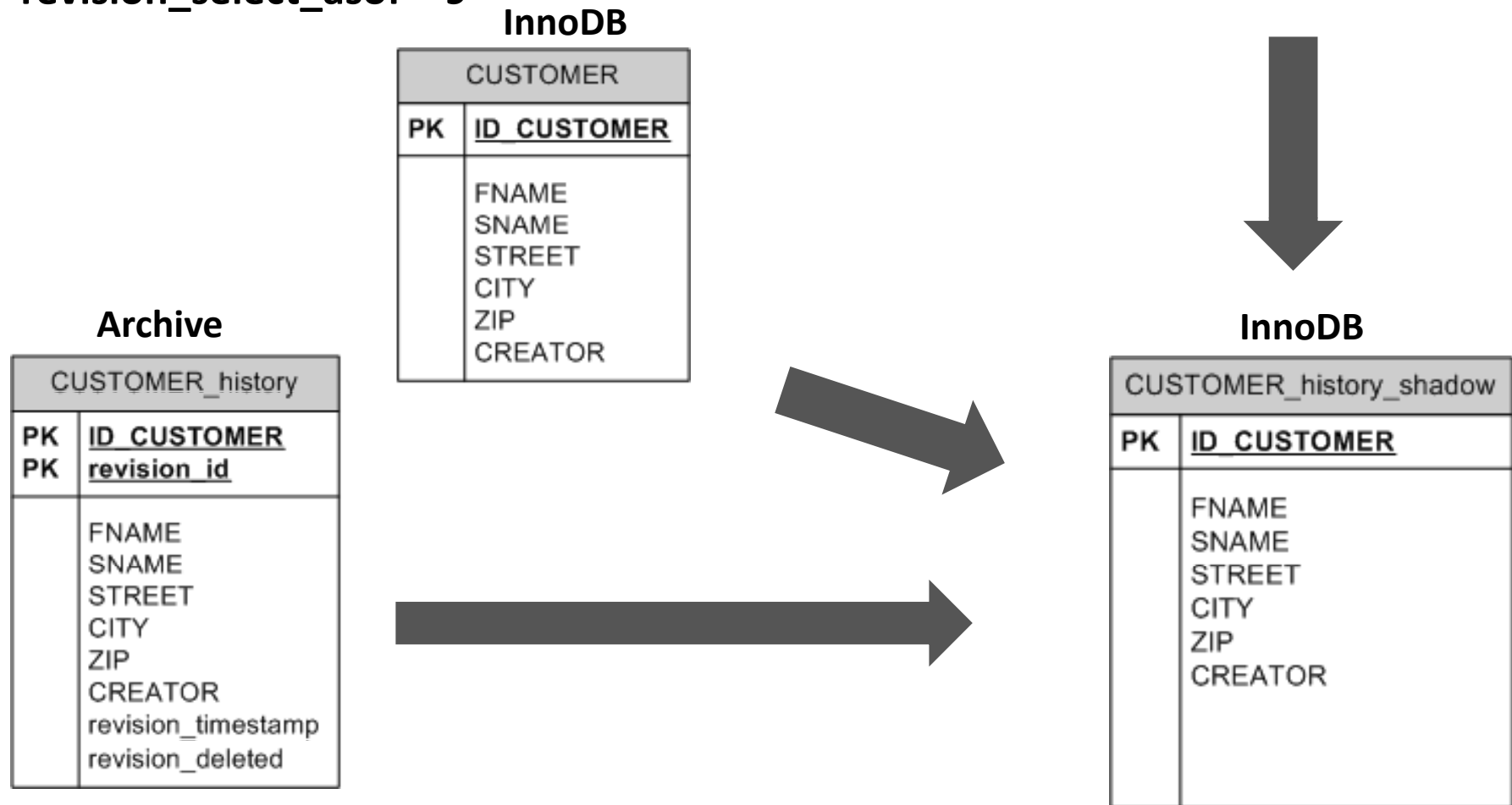
Permanent transformation between versioning table types expensive (drop old table, create new table, insert data)

Transformations for single select-TX or transaction sequence

*If **session** performs select-TX on history data create shadow table with **session specific** data*

Shadow table

revision_select_mode = version
revision_select_asof = 9



Challenges

“Auto” is optimal for insert and update performance via archive engine

Costs for table transformation once per session

Shadow table share between sessions?

How many selects before shadow is created?

Temporarily permanent shadow tables?

Incremental shadow tables?

V0.1 – Alpha released Sep '08

- Basic versioning
- Homogenous storage engines
- Multiple versioning schemas

V0.2 – Beta release '09

- Bugfixes (thanks Giuseppe)
- Hybrid storage engines
- Insertion into history table as “admin”
- Column fine versioning

VX

- Intermediate optimizer for history tables (table scan vs. index)
- Temporary indexes for shadow tables
- Automated schema selection
- As of syntax for select modes
- Automated storage engine selection

Conclusion

Transparent versioning for table data

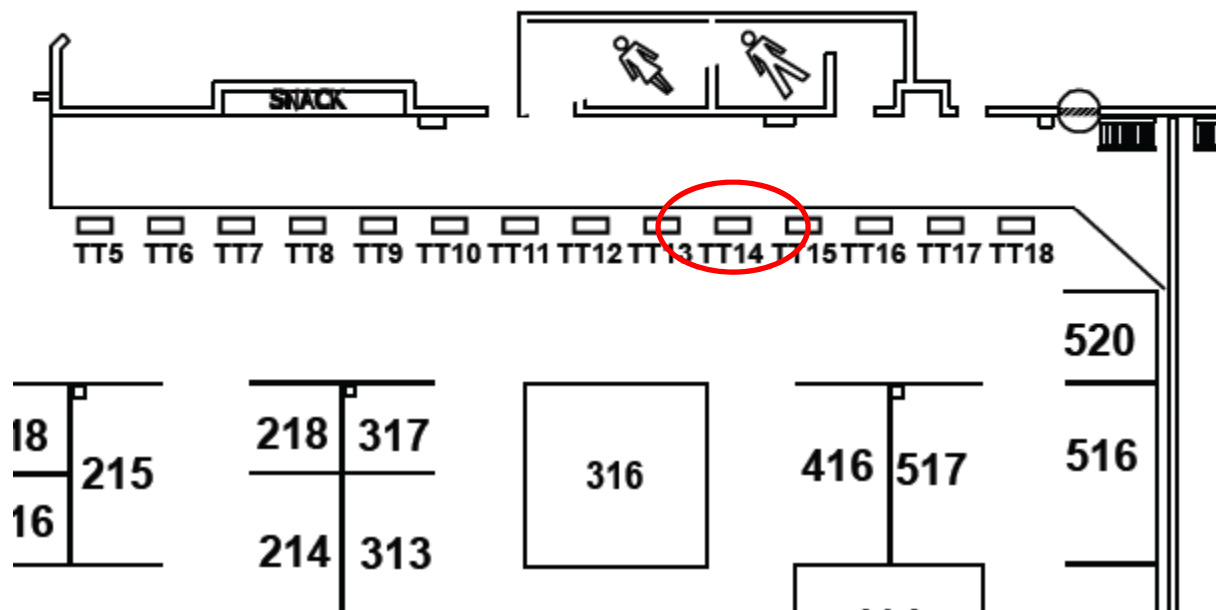
Usage of existing SE as base table

Storage of data versions with different schemas

Disk optimal and SELECT optimal schemas

Transformation of history tables into shadow tables for fast ASOF syntax

Thanks to
 Guillaume Delannoy
 Wolfgang Lehner (Technische Universität Dresden)



Automated data versioning

Peter Benjamin Volk^{* +}, Guillaume Delannoy^{*}

^{*}DDEngine.org

⁺Technische Universität Dresden
Database Technology Group