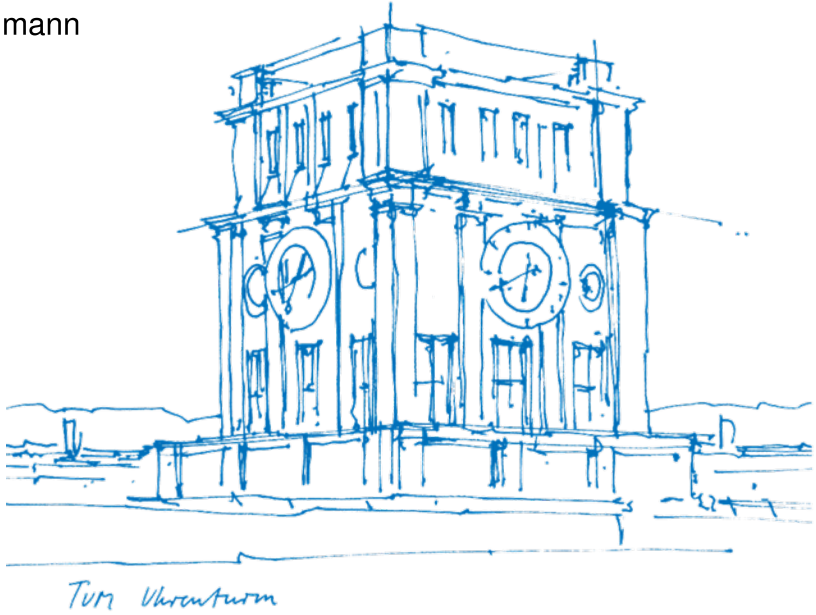
Chair III: Database Systems
Chair XXV: Data Science and Engineering
Department of Informatics
Technical University of Munich

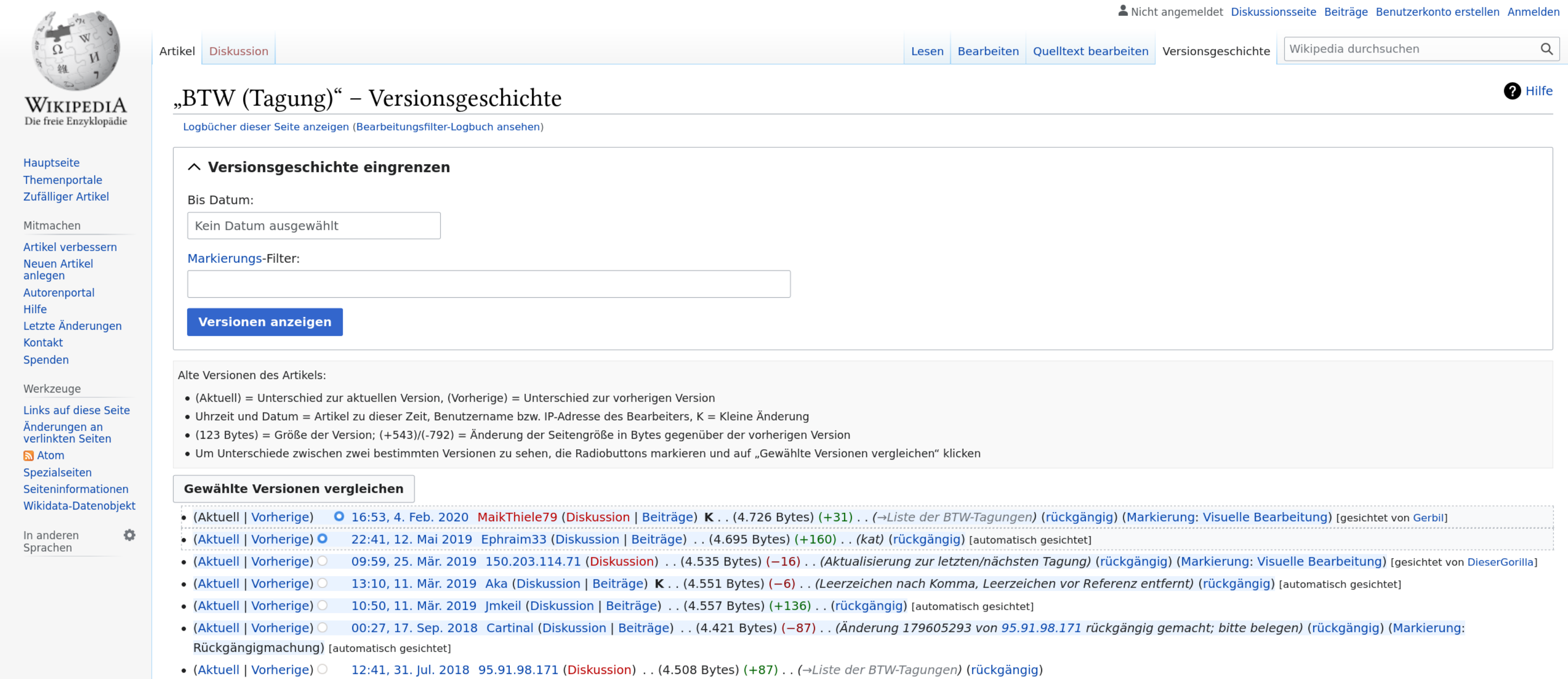
Umbra as a Time Machine: Adding a Versioning Type to SQL

Lukas Karnowski, Maximilian E. Schüle, Alfons Kemper, Thomas Neumann

Dresden, Germany, May 10, 2021



Wikipedia: Version Control with Meta Tables



Nicht angemeldet Diskussionsseite Beiträge Benutzerkonto erstellen Anmelden

Artikel Diskussion | Lesen Bearbeiten Quelltext bearbeiten Versionsgeschichte | Wikipedia durchsuchen

Wikipedia
Die freie Enzyklopädie

Hauptseite
Themenportale
Zufälliger Artikel

Mitmachen
Artikel verbessern
Neuen Artikel anlegen
Autorenportal
Hilfe
Letzte Änderungen
Kontakt
Spenden

Werkzeuge
Links auf diese Seite
Änderungen an verlinkten Seiten
Atom
Spezialseiten
Seiteninformationen
Wikidata-Datenobjekt

In anderen Sprachen

„BTW (Tagung)“ – Versionsgeschichte

Hilfe

Logbücher dieser Seite anzeigen (Bearbeitungsfilter-Logbuch ansehen)

Versionsgeschichte eingrenzen

Bis Datum:
Kein Datum ausgewählt

Markierungs-Filter:

Versionen anzeigen

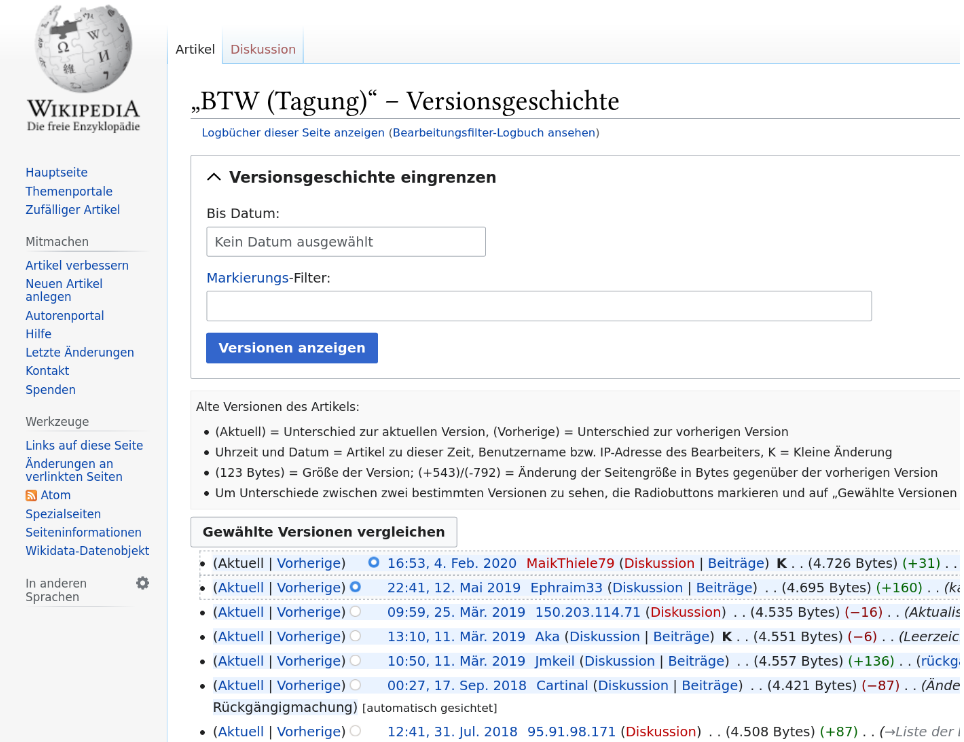
Alte Versionen des Artikels:

- (Aktuell) = Unterschied zur aktuellen Version, (Vorherige) = Unterschied zur vorherigen Version
- Uhrzeit und Datum = Artikel zu dieser Zeit, Benutzername bzw. IP-Adresse des Bearbeiters, K = Kleine Änderung
- (123 Bytes) = Größe der Version; (+543)/(-792) = Änderung der Seitengröße in Bytes gegenüber der vorherigen Version
- Um Unterschiede zwischen zwei bestimmten Versionen zu sehen, die Radiobuttons markieren und auf „Gewählte Versionen vergleichen“ klicken

Gewählte Versionen vergleichen

- (Aktuell | Vorherige) 16:53, 4. Feb. 2020 MaikThiele79 (Diskussion | Beiträge) **K** . . (4.726 Bytes) (+31) . . *(→Liste der BTW-Tagungen)* (rückgängig) (Markierung: Visuelle Bearbeitung) [gesichtet von Gerbil]
- (Aktuell | Vorherige) 22:41, 12. Mai 2019 Ephraim33 (Diskussion | Beiträge) . . (4.695 Bytes) (+160) . . *(kat)* (rückgängig) [automatisch gesichtet]
- (Aktuell | Vorherige) 09:59, 25. Mär. 2019 150.203.114.71 (Diskussion) . . (4.535 Bytes) (−16) . . *(Aktualisierung zur letzten/nächsten Tagung)* (rückgängig) (Markierung: Visuelle Bearbeitung) [gesichtet von DieserGorilla]
- (Aktuell | Vorherige) 13:10, 11. Mär. 2019 Aka (Diskussion | Beiträge) **K** . . (4.551 Bytes) (−6) . . *(Leerzeichen nach Komma, Leerzeichen vor Referenz entfernt)* (rückgängig) [automatisch gesichtet]
- (Aktuell | Vorherige) 10:50, 11. Mär. 2019 Jmkeil (Diskussion | Beiträge) . . (4.557 Bytes) (+136) . . (rückgängig) [automatisch gesichtet]
- (Aktuell | Vorherige) 00:27, 17. Sep. 2018 Cartinal (Diskussion | Beiträge) . . (4.421 Bytes) (−87) . . *(Änderung 179605293 von 95.91.98.171 rückgängig gemacht; bitte belegen)* (rückgängig) (Markierung: Rückgängigmachung) [automatisch gesichtet]
- (Aktuell | Vorherige) 12:41, 31. Jul. 2018 95.91.98.171 (Diskussion) . . (4.508 Bytes) (+87) . . *(→Liste der BTW-Tagungen)* (rückgängig)

Wikipedia: Version Control with Meta Tables




```sql
CREATE TABLE page (
  page_id INT PRIMARY KEY,
  page_title TEXT,
  page_latest INT REFERENCES pagecontent (old_id)
);
CREATE TABLE revision (
  rev_id INT PRIMARY KEY,
  rev_page INT REFERENCES page (page_id),
  rev_text_id INT REFERENCES pagecontent (old_id),
  rev_parent_id INT,
  rev_timestamp TIMESTAMP
);
CREATE TABLE pagecontent (
  old_id INT PRIMARY KEY,
  old_text TEXT
);
```

	Size	Compression
Full Page Edit History	35.0 GiB	-
Current Version Only	1.1 GiB	-
History as File Diffs	14.0 GiB	59.77 %
History as Edit Diffs	9.4 GiB	72.71 %

Table: Estimation of saved storage when using compression techniques based on the *Simple English* Wikipedia page edit history dump of October 1, 2018.

Background: Wikipedia Articles for Benchmarking

Monopedia (VLDB 2017)

- single main-memory database server (HyPer) sufficient for web scale applications (full English Wikipedia)
- Monopedia Benchmark: simulate load of Wikipedia

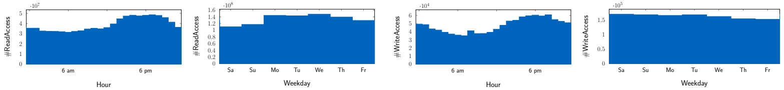
Monopedia: Staying Single is Good Enough

The HyPer Way for Web Scale Applications

Maximilian E. Schüle, Pascal M. N. Schliski, Thomas Hutzelmann, Tobias Rosenberger
Viktor Leis, Dimitri Vorona, Alfons Kemper, Thomas Neumann
{m.schuele,p.schliski,t.hutzelmann,tobias.rosenberger}@tum.de, {leis,vorona,kemper,neumann}@in.tum.de

	#req	traffic [MiB]	#err	req/s	KiB/s	On Time	min	avg	max	last request	title	Rate
Σ Reads	2705	64782007	0	496.9	12766...	91.26%	803	3297.97	298488	3826		
Σ Writes	7	171728	0	0.0	0.0	100.00%	1426	64192....	403219	6487		
R0	2202	53455458	0	332.9	88998...	82.54%	803	3221.30	298488	3759	Isla_Fisher	
R1	503	11326549	0	164.0	38665...	99.99%	987	3374.64	298468	3826	Pope_Sergius_I	
W2	7	171728	0	0.0	0.0	100.00%	1426	64192....	403219	6487	A_Tale_of_Two_C...	

Interactive webinterface of Monopedia Benchmark: summarized informations and informations for each worker thread (request frequency, duration, timeliness, name of last article accessed); rate can be adjusted.



Traffic Data Analysis: March 25-31, 2017: aggregated by week and hour to find out the traffic peaks and the time-dependent load distribution; more read requests during weekdays; nearly constant number of updates. left: `SELECT SUM(reads) FROM week WHERE date BETWEEN '2017/03/25' AND '2017/03/31'`, right: `SELECT SUM(writes) FROM week WHERE data BETWEEN '2017/03/25' AND '2017/03/31'`

TardisBenchmark/TardisDB (SSDBM 2019, SIGMOD 2021)

- integrate versioning of tables into database systems
- benchmark performance (time/space) using Wikipedia articles

Versioning in Main-Memory Database Systems

From MusaeusDB to TardisDB

Maximilian E. Schüle
maximilian.schuele@tum.de
Lukas Karnowski
lukas.karnowski@tum.de
Josef Schmeißer
josef.schmeisser@tum.de
Benedikt Kleiner
benedikt.kleiner@tum.de
Alfons Kemper
kemper@in.tum.de
Thomas Neumann
neumann@in.tum.de
Technical University of Munich

ABSTRACT

As relational database systems do not support collaborative dataset editing, online lexicons—such as Wikipedia's MediaWiki—build their own version control above the database system to allow constraint-preserving version checkouts or commits involving multiple tables. To eliminate the need for purpose-specific solutions, we propose adding version control as a layer on top of the database system or integrating versioning in the database system's core.

This paper presents the first two architectures for versioning an entire state of a database system with respect to references among multiple relations. We design the prototype *MusaeusDB* as a solution for existing database systems, either as an external tool or as an extended SQL interface. The prototype *TardisDB*—an extended main-memory database system—reuses multi-version concurrency control for in-place updates while keeping older versions accessible. For performance tests on different storage layouts, we create—based on Wikipedia's page history—the *TardisBenchmark*. Our results show that it is indeed feasible to reduce wasted space while still ensuring constant retrieval time. Also, extending a main-memory database system's multi-version concurrency control has no negative impact on the transactional throughput. For further research

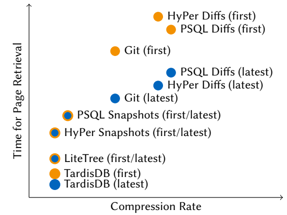


Figure 1: Sketch of the trade-off between storage savings (compression rate) and retrieval time: storing only one version snapshot and computing the others out of the changed differences (diffs) will reduce the amount of storage needed but will increase the retrieval time.

Space Compression

- **Question:** How to reduce the space consumption of Wikipedia articles within a database system?

Requirements for a Versioning System

$\rightarrow \mid \leftarrow$	Δ	$\rightarrow$	ACID/MVCC	SQL
compressing full articles	Enable delta compression	Efficient retrieval times (focus on latest version)	Database system guarantees	SQL as the declarative programming language

```
CREATE TABLE page (page_id INT PRIMARY KEY, page_title TEXT, old_text DIFFTEXT);
```

Space Compression through `DiffText` Datatype

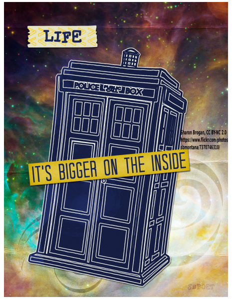
- **Question:** How to reduce the space consumption of Wikipedia article within a database system?
- **Solution:** SQL datatype, that compresses text by avoiding redundancy

```
CREATE TABLE wikidiff (title text, content difftext);
INSERT INTO wikidiff (SELECT 'example', BUILD('first', 'first␣version', 'second␣version'));
SELECT GET_CURRENT_VERSION(difftext) FROM wikidiff;
```

UMBRA

Umbra Integration

SQL Integration



Evaluation

Umbra Integration

Umbra Integration

$D_{T_2 \rightarrow T_1}$	$D_{T_3 \rightarrow T_2}$	$\mathbf{T_3}$	$D_{T_5 \rightarrow T_4}$	$\mathbf{T_5}$

Figure: Chain of diffs, every third version a complete snapshot (bold).

- Umbra: code-generating database system with in-memory performance
- offers SQL datatype for flexible sized strings
- DiffText datatype based on this datatype for variable length data
- latest version as a snapshot followed by backward deltas
- snapshots inbetween for efficient retrieval time

Umbra Integration

- Umbra: code-generating database system with in-memory performance
- offers SQL datatype for flexible sized strings
- DiffText datatype based on this datatype for variable length data
- latest version as a snapshot followed by backward deltas
- snapshots inbetween for efficient retrieval time

$D_{T_2 \to T_1}$	$D_{T_3 \to T_2}$	$\mathbf{T_3}$	$D_{T_5 \to T_4}$	$\mathbf{T_5}$

Figure: Chain of diffs, every third version a complete snapshot (bold).

```
struct DiffTextRepresentation {
   uint32_t currentOffset;     // Offset of current version in data section
   uint32_t currentLength;     // Length of current version
   uint32_t arraySize;         // The size of the version pointer's array
   uint16_t diffsToFullCount;  // Counter of diffs until next full version
   struct {                    // Array of pairs, pointing into the data section
      uint32_t offset;         // Offset of version in data section
      bool full;               // Is this a full version?
      uint32_t patchStart;     // Start of patch
      uint32_t patchEnd;       // End of patch
   } versionPointers[];
   // Data section follows this struct immediately
};
```

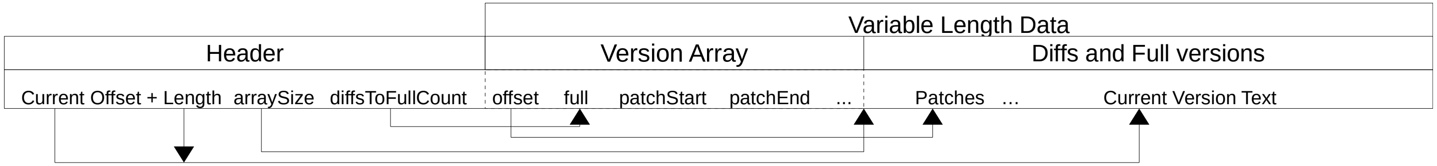


Figure: Structure of a DiffText tuple.

SQL Integration

SQL Integration: Operations

- `BUILD(`$T_1, \ldots, T_N$`)`: creation of a DiffText object out of N versions.
- `APPEND(`$D, T_1, \ldots, T_N$`)`: appending N versions to an existing object.
- `SET_CURRENT_VERSION(`D, T`)`: adding a new version, equal to `APPEND(`D, T`)`.
- `GET_VERSION_BY_ID(`D, N`)`: extract version N.
- `GET_CURRENT_VERSION(`D`)` returns the latest version, equal to `GET_VERSION_BY_ID(`D, N`)` (#Versions=N).
- `EXPAND(`D, M, N): unary database operator for extracting versions M to N.

SQL Integration: Example

```
CREATE TABLE example (value DiffText); INSERT INTO example (SELECT BUILD('first'));
```

currentOffset	0
currentLength	5
arraySize	0
diffsToFullCount	0

Data	First

SQL Integration: Example

```
CREATE TABLE example (value DiffText); INSERT INTO example (SELECT BUILD('first'));
UPDATE example SET value=SET_CURRENT_VERSION(value,'first version');
```

currentOffset	0
currentLength	13
arraySize	1
diffsToFullCount	1

offset	0
full	*false*
patchStart	5
patchEnd	13

Data	First_Version

SQL Integration: Example

```
CREATE TABLE example (value DiffText); INSERT INTO example (SELECT BUILD('first'));
UPDATE example SET value=SET_CURRENT_VERSION(value,'first version');
UPDATE example SET value=SET_CURRENT_VERSION(value,'second version');
```

currentOffset	5
currentLength	14
arraySize	2
diffsToFullCount	2
offset	0
full	*false*
patchStart	5
patchEnd	13
offset	0
full	*false*
patchStart	0
patchEnd	6
Data	FirstSecond_Version

SQL Integration: Example

```
INSERT INTO example (SELECT BUILD('first', 'first␣version', 'second␣version'));
```

currentOffset	5
currentLength	14
arraySize	2
diffsToFullCount	2
offset	0
full	*false*
patchStart	5
patchEnd	13
offset	0
full	*false*
patchStart	0
patchEnd	6
Data	FirstSecond_Version 0123456789012345678

```
>SELECT GET_CURRENT_VERSION(value) FROM example;
Second_Version
```

SQL Integration: Example

```
INSERT INTO example (SELECT BUILD('first', 'first␣version', 'second␣version'));
```

currentOffset currentLength arraySize diffsToFullCount	5 14 2 2
offset full patchStart patchEnd	0 *false* 5 13
offset full patchStart patchEnd	0 $\Rightarrow$ Patch: $[0,5)$ *false* 0+5 6+5
Data	First~~Second~~_Version 0123456789012345678

```
>SELECT GET_CURRENT_VERSION(value) FROM example;
Second_Version
 >SELECT GET_VERSION_BY_ID(value,2) FROM example;
First_Version
```

SQL Integration: Example

```
INSERT INTO example (SELECT BUILD('first', 'first version', 'second version'));
```

currentOffset currentLength arraySize diffsToFullCount	5 14 2 2
offset full patchStart patchEnd	0 ⇒ Patch: [0,0) *false* 5 13+6
offset full patchStart patchEnd	0 *false* 0 6
Data	First~~Second_Version~~ 0123456789012345678

```
>SELECT GET_CURRENT_VERSION(value) FROM example;
Second_Version
 >SELECT GET_VERSION_BY_ID(value,2) FROM example;
First_Version
 >SELECT GET_VERSION_BY_ID(value,1) FROM example;
First
```

Evaluation


POLICE PUBLIC CALL BOX
Sharon Brogan, CC BY-NC 2.0
https://www.flickr.com/photos/
sbmontana/7370746310/
IT'S BIGGER ON THE INSIDE

Evaluation: Space Consumption

- **System:** Ubuntu 18.04 LTS, Intel Xeon CPU E5-2660 v2 processor, 2.20 GHz (20 cores), 256 GiB DDR4 RAM
- **Data:** Wikipedia dumps from 09/01/2019 (pages 971896 to 972009), Size of XML file: about `120` MB
- **Observation:** Good compression when storing every 20th version as a snapshot

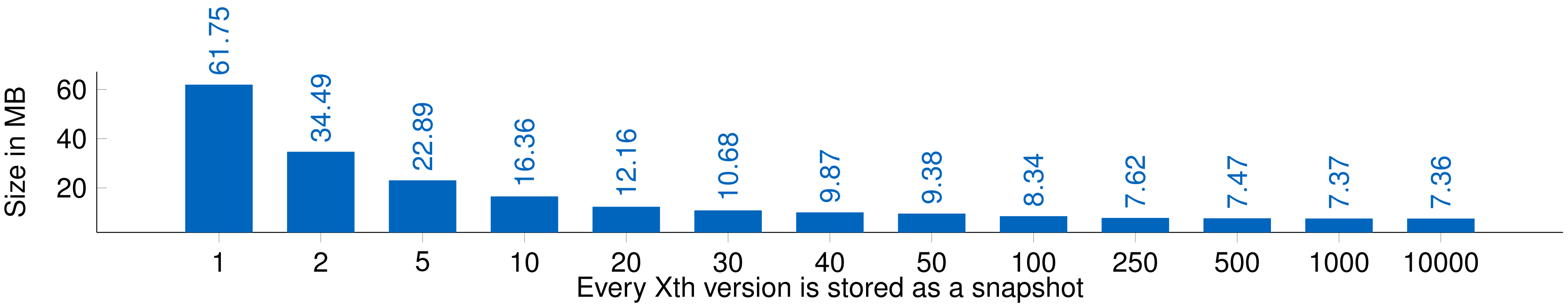


Figure: Memory consumption depending on the frequency of stored snapshots.

Evaluation: `X=50`

Listing 1: Diff

```
INSERT INTO t (text) VALUES (BUILD(T1, ..., TN));
SELECT EXPAND(text, 1, N) from t;
```

Listing 2: Snapshot

```
INSERT INTO t (rev_id, text) VALUES (1, T1), ..., (N, TN);
SELECT text from t;
```

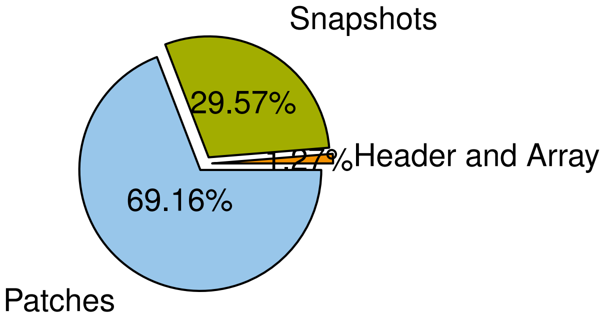


Figure: Fractions.

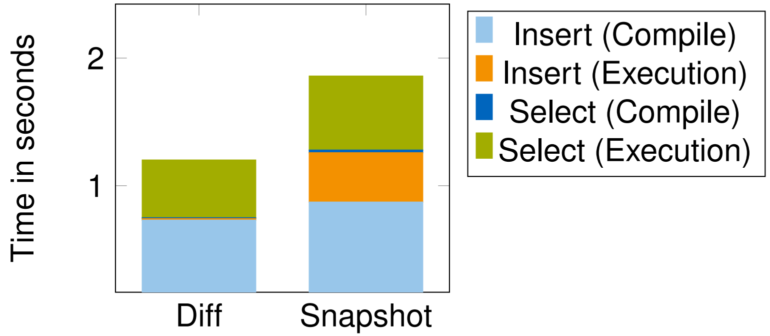


Figure: Each version as a single snapshot or in one DiffText object.

- **Observation:** DiffText datatype faster for insertion and retrieval as less operations are required.
- Parsing time for text input part of compile time for insertion.

Conclusion and Future Work

```
CREATE TABLE wikidiff (title text, content difftext);
INSERT INTO wikidiff (SELECT 'example', BUILD('first', 'first␣version', 'second␣version'));
SELECT GET_CURRENT_VERSION(difftext) FROM wikidiff;
```

Conclusion

- SQL text data type for compression
- focus: retrieval times, not build time
- achieved compression rate of about 88 % (62 MB each snapshot within the database, 7.36 MB when storing the differences for $X = 10000$)

Future Work

- investigate on algorithms with stronger compression
- optimise build times
- store older versions on background memory
- combine with table versioning

Thank you for your attention!