Exploiting string compression in data systems

Peter Boncz

+Viktor Leis, Thomas Neumann, Tim Gubner,
Bogdan Ghita, Diego Tome

CWI
Database Architectures

DBtest @ SIGMOD18

Get Real: How Benchmarks Fail to Represent the Real World

Adrian Vogelsgesang, Michael Haubenschild,
Jan Finis, Alfons Kemper, Viktor Leis, Tobias Muehlbauer, Thomas Neumann, Manuel Then

Tableau Software

{avogelsgesang, mhaubenschild, jfinis, akemper, vleis, tmuehlbauer, tneumann, mthen}@tableau.com

ABSTRACT

Industrial as well as academic analytics systems are usually evaluated based on well-known standard benchmarks, such as TPC-H or TPC-DS. These benchmarks test various components of the DBMS including the join optimizer, the implementation of the join and aggregation operators, concurrency control and the scheduler. However, these benchmarks fall short of evaluating the "real" challenges imposed by modern BI systems, such as Tableau, that emit machine-generated query workloads. This paper reports a comprehensive study based on a set of more than 60k real-world BI data repositories together with their generated query workload. The machine-generated workload posed by BI tools differs from the "hand-crafted" benchmark queries in multiple ways: Structurally simple relational operator trees often come with extremely complex scalar expressions such that expression evaluation becomes the limiting factor. At the same time, we also encountered much more complex relational operator trees than covered by benchmarks. This long tail in both, operator tree and expression complexity, is not adequately represented in standard benchmarks. We contribute various statistics gathered from the large dataset, e.g., data type distributions, operator frequency, string length distribution and

3 DATASETS

In this section we focus on the dataset characteristics before delving into the query workload characteristics in Sec. 4.

3.1 Strings are Everywhere

The TPC-H benchmark uses integer keys for all relations. In contrast, our real-world dataset mostly features strings as keys: ISO country codes are used to identify countries, IANA codes for airports and ISBNs for books. UUIDs or other alphanumeric identifiers are also common choices where pre-established keys are not available. All those different flavors of "surrogate" keys have one thing in common: They are stored as strings in the database, i.e., either as `VARCHAR`, `CHAR` or `TEXT` depending on the DBMS and administrator.

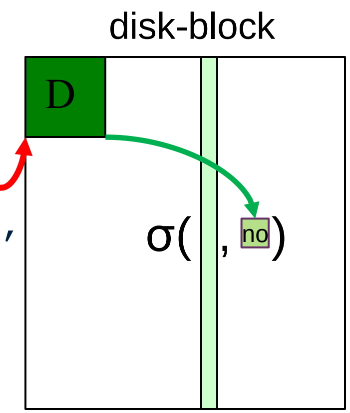
Similarly, other non-key columns that a DBA would normally specify as `INTEGER` or even as a boolean are also commonly stored as strings. Our dataset shows that more than 60% of the single-character strings are `0` or `1`. With a combined frequency of 4.5%, the characters "Y" and "N" are also very popular to represent "yes" and "no", respectively. In a cleanly designed schema, those columns would be represented as booleans. Another common pattern is to store fiscal years as strings in the form of "2017/18". In general, while

String Compression in a DBMS

- Dictionary Compression
  - Whole string becomes 1 code, points into a dictionary D
  - works well if there are few unique strings (many repetitions)

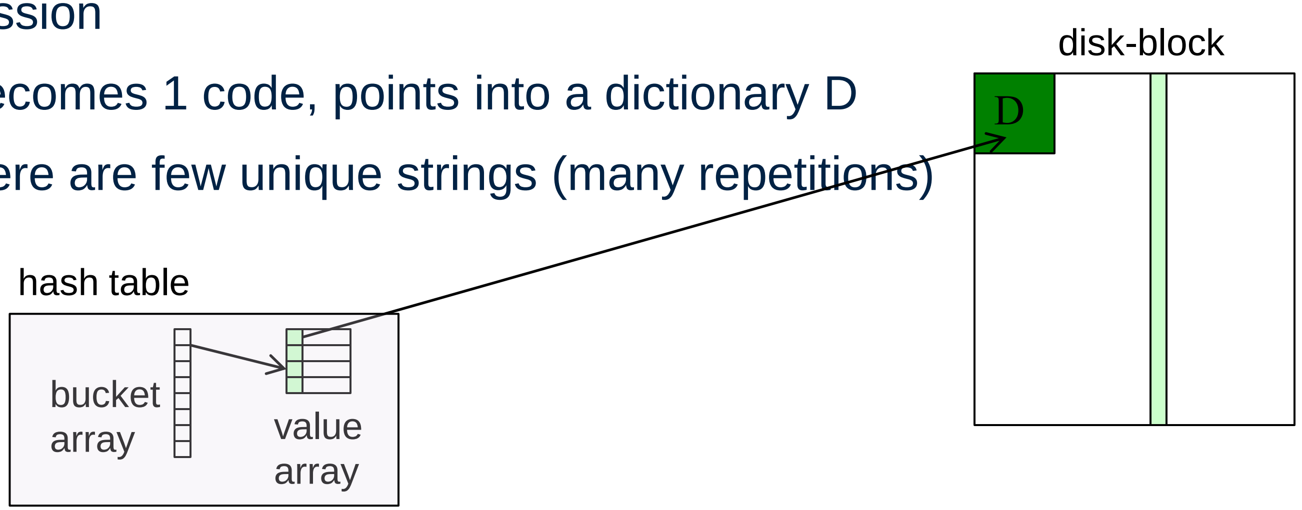
String Compression in a DBMS

- Dictionary Compression
  - Whole string becomes 1 code, points into a dictionary D
  - works well if there are few unique strings (many repetitions)
  - Allows **predicate pushdown** `select * from T where S='no'`



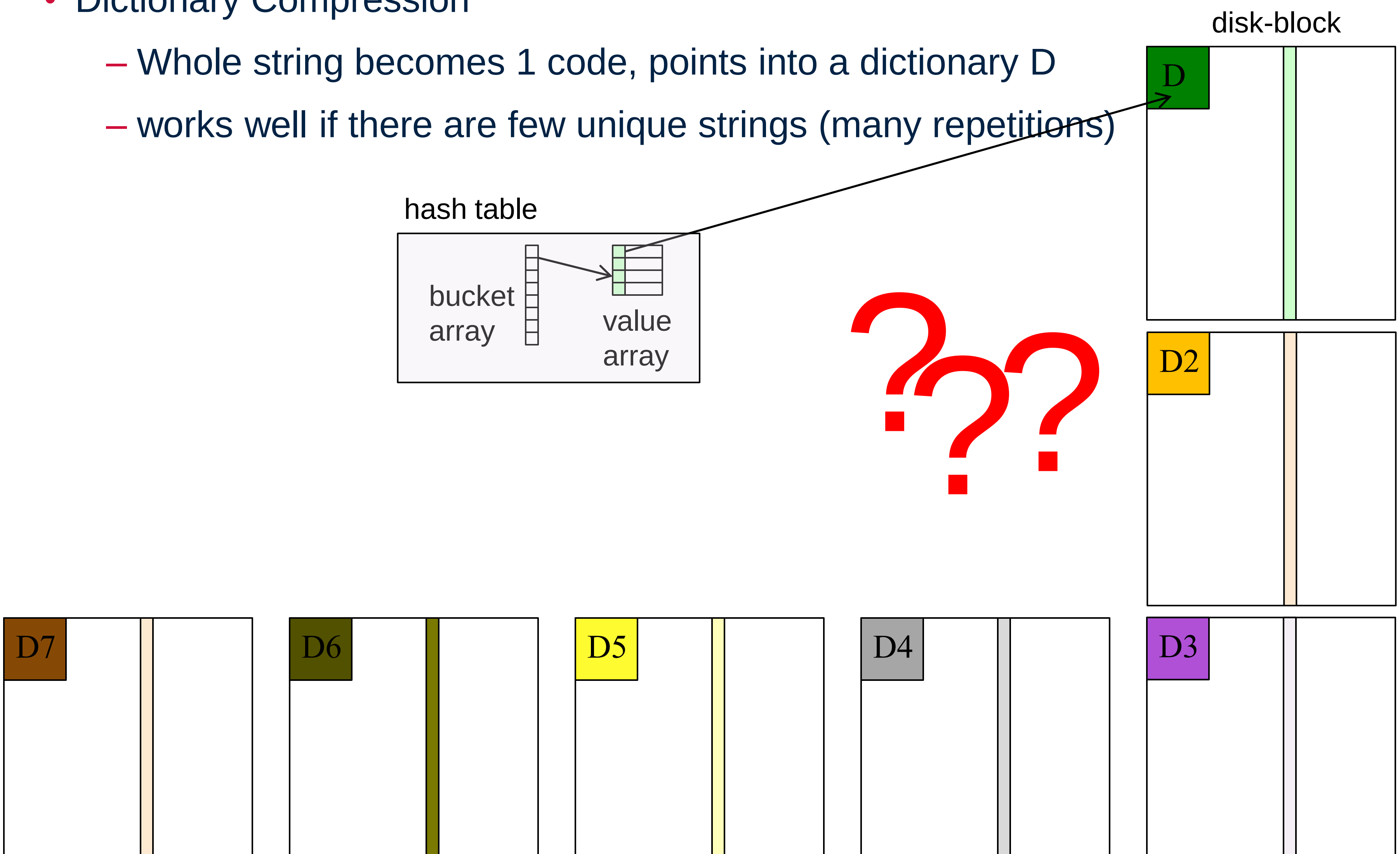
String Compression in a DBMS

- Dictionary Compression
  - Whole string becomes 1 code, points into a dictionary D
  - works well if there are few unique strings (many repetitions)



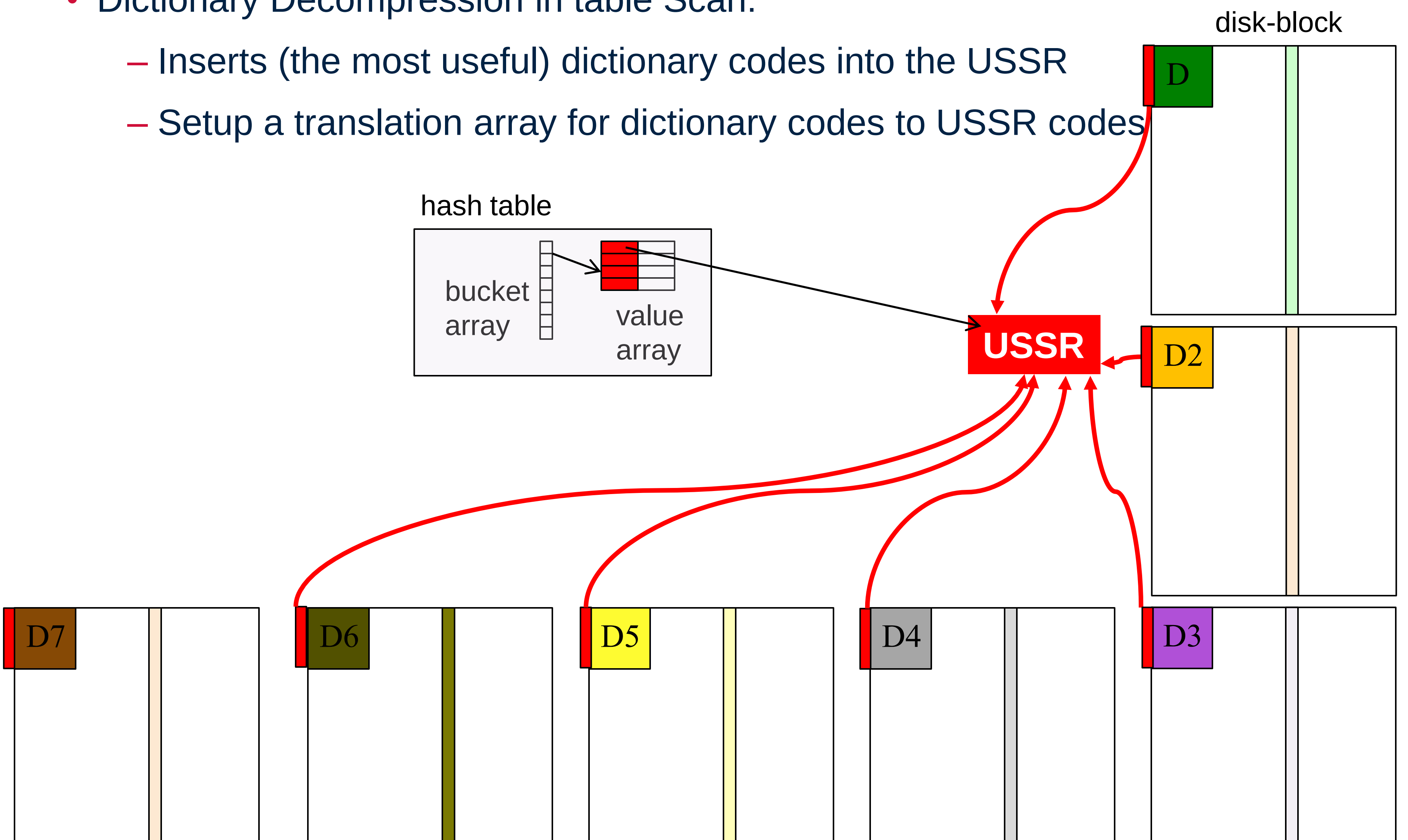
String Compression in a DBMS

- Dictionary Compression
  - Whole string becomes 1 code, points into a dictionary D
  - works well if there are few unique strings (many repetitions)



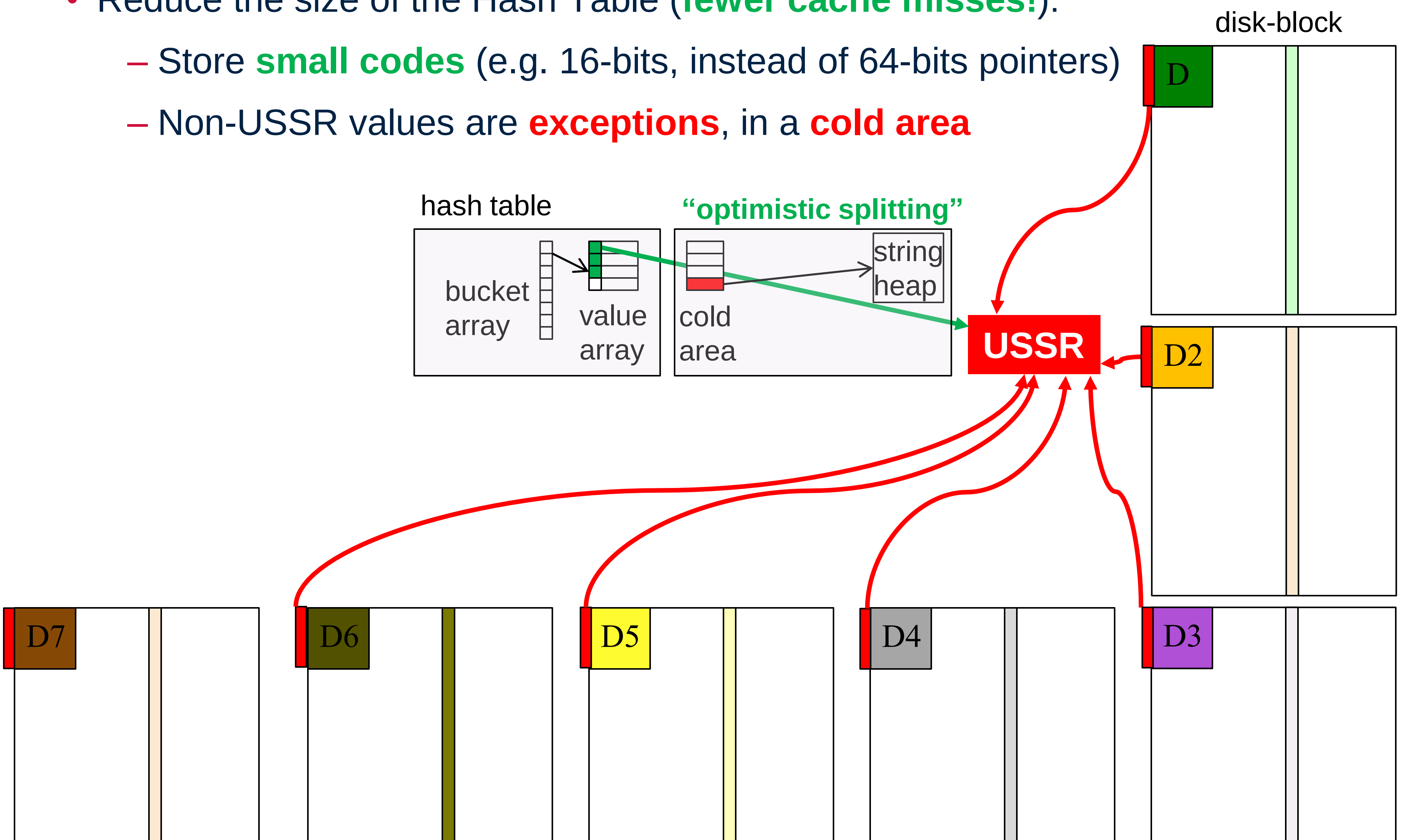
Unique Self-aligned String Region (USSR)

- Dictionary Decompression in table Scan:
  - Inserts (the most useful) dictionary codes into the USSR
  - Setup a translation array for dictionary codes to USSR codes



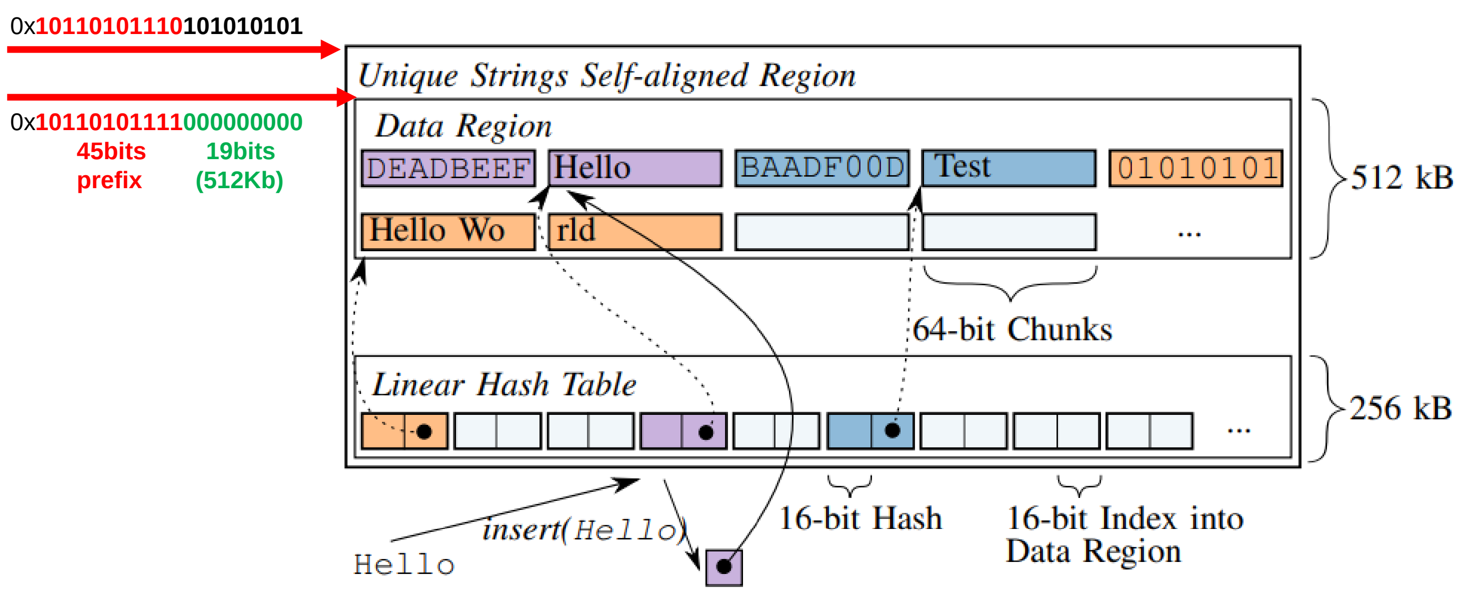
Optimistic Splitting

- Reduce the size of the Hash Table (**fewer cache misses!**):
  - Store **small codes** (e.g. 16-bits, instead of 64-bits pointers)
  - Non-USSR values are **exceptions**, in a **cold area**



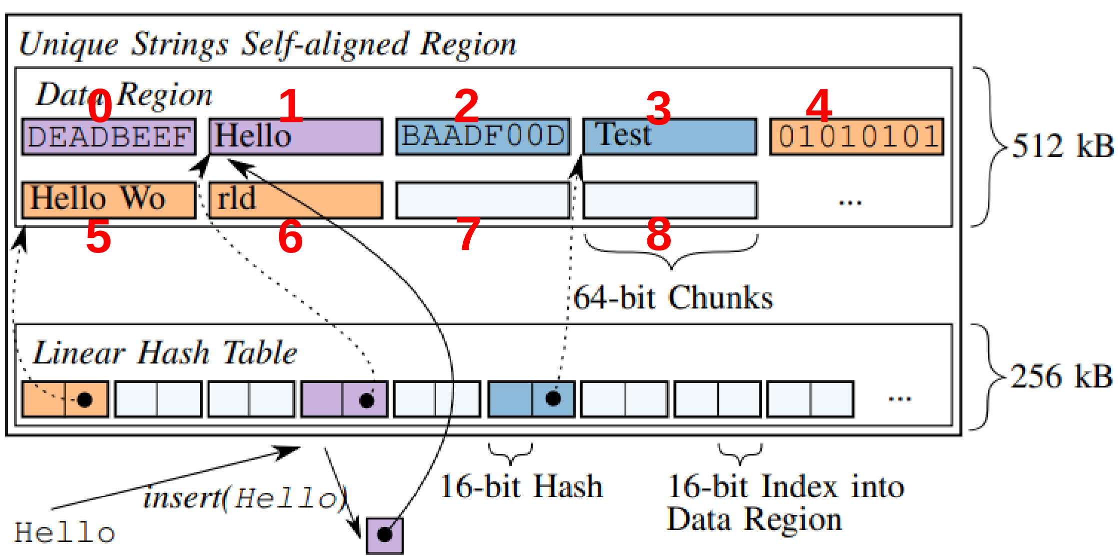
USSR inner Workings

- Gather a **global** dictionary **on-the-fly** *only valid for the query*
- A small (cache-resident) area: only the most useful data
  - The area has a special property: it is a **self-aligned** memory pointer
    - All the pointers into it start the same (have the same bit prefix)
    - USSR strings are recognizable **quickly** by their pointer
    - Fast **linear hash table** for very quick inserts of new strings on-the-fly



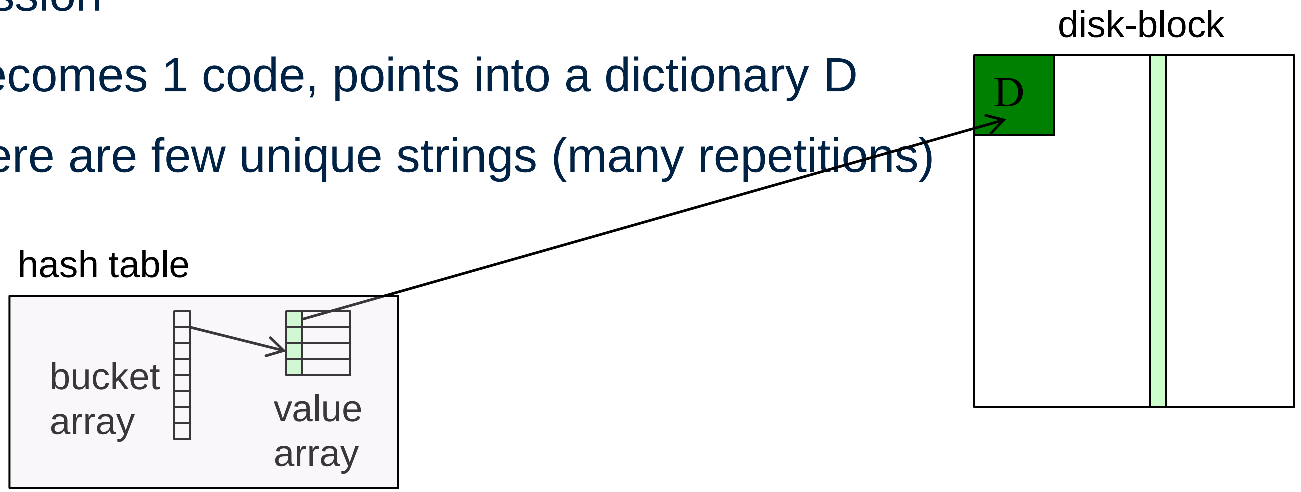
USSR inner workings

- Gather a **global** dictionary **on-the-fly**
- A small (cache-resident) area: only the most useful data
  - The area has a special property: it is a **self-aligned** memory pointer
  - The strings in the USSR are **aligned** to eg 8 byte multiples
    - precompute the hash (length is part of it), store it before the pointer
    - you can identify each USSR string by a small **slot number** (16 bits)



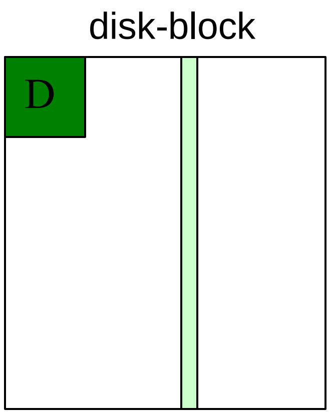
String Compression in a DBMS

- Dictionary Compression
  - Whole string becomes 1 code, points into a dictionary D
  - works well if there are few unique strings (many repetitions)

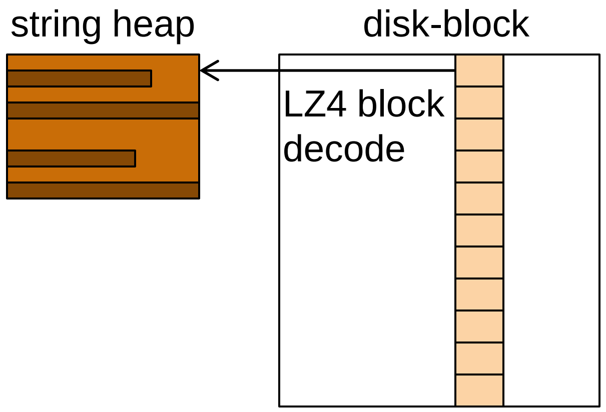


String Compression in a DBMS

- Dictionary Compression
  - Whole string becomes 1 code, points into a dictionary D
  - works well if there are few unique strings (many repetitions)

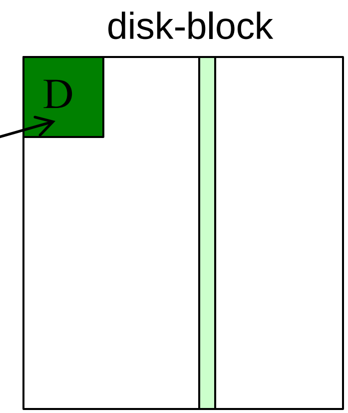


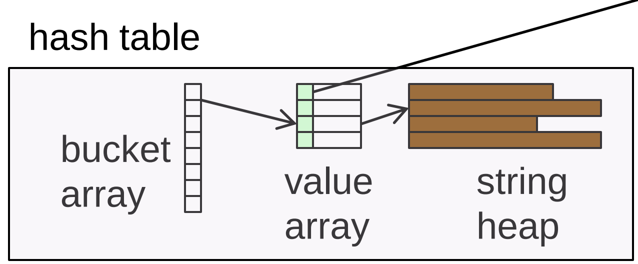
- Heavy-weight/general-purpose Compression
  - Lempel-Zipf plus possibly entropy coding
  - Zip, gzip, snappy, **LZ4**, zstd, …
  - Block-based decompression



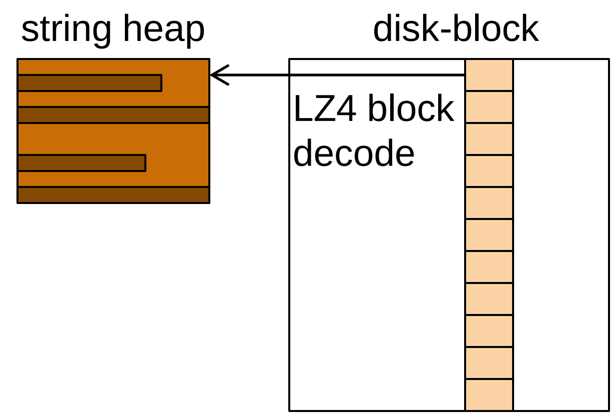
String Compression in a DBMS

- Dictionary Compression
  - Whole string becomes 1 code, points into a dictionary D
  - works well if there are few unique strings (many repetitions)



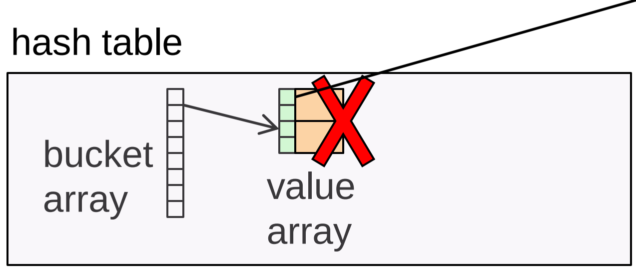


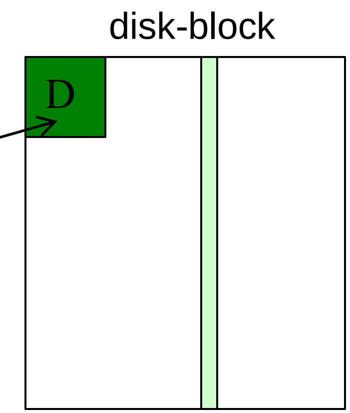
- Heavy-weight/general-purpose Compression
  - Lempel-Zipf plus possibly entropy coding
  - Zip, gzip, snappy, **LZ4**, zstd, …
  - Block-based decompression



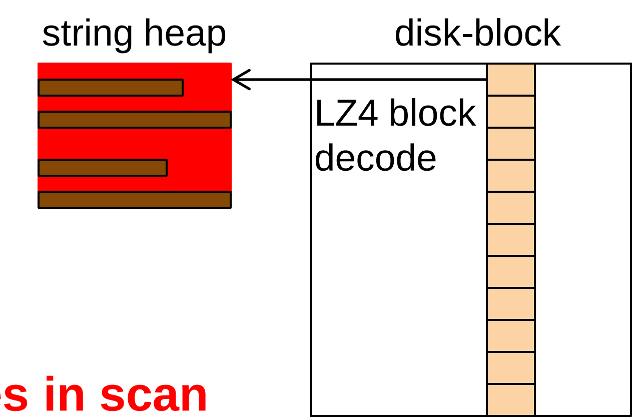
String Compression in a DBMS

- Dictionary Compression
  - Whole string becomes 1 code, points into a dictionary D
  - works well if there are (relatively) few unique strings



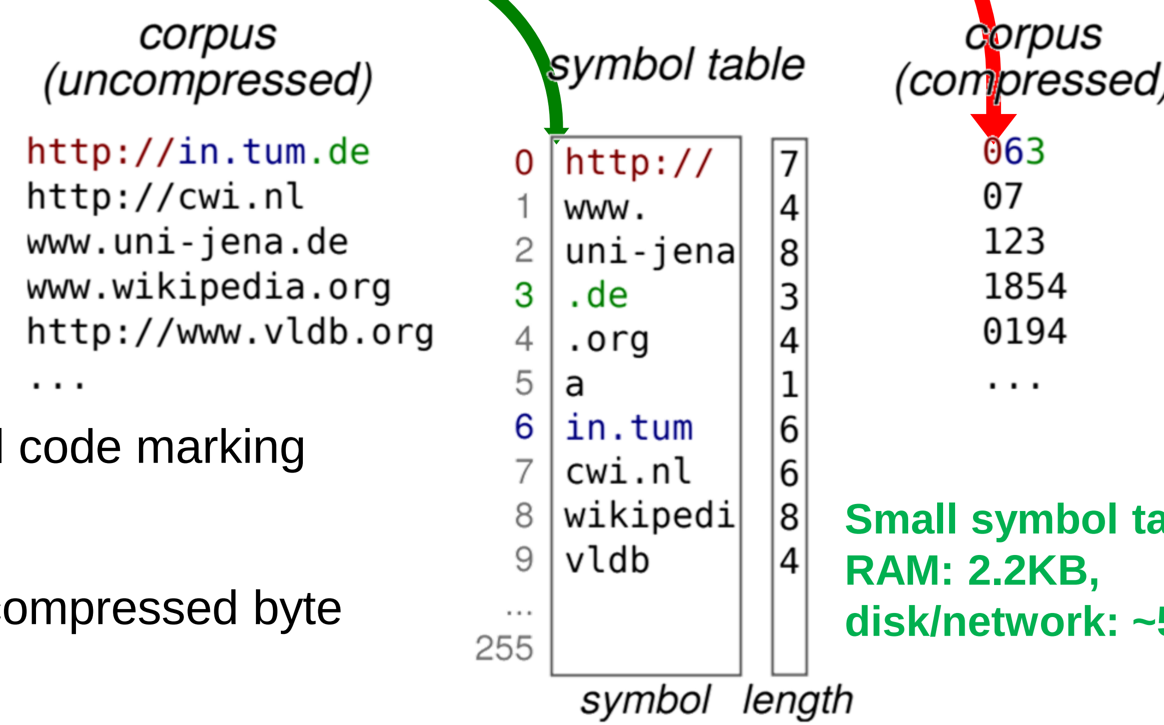


- Heavy-weight/general-purpose Compression
  - Lempel-Zipf plus possibly entropy coding
  - Zip, gzip, snappy, **LZ4**, zstd, …
  - Block-based decompression
    - **must decompress (all=) unneeded values in scan**
    - **cannot be leveraged in hash tables, sorting, network shuffles**
    - **FSST targets compression of many small textual strings**



FSST: Fast Static Symbol Table string compression

- Encode strings as a sequence of bytes, where each byte [0,254] is a
  - **CODE**
- Each code stands for a 1-8 byte
  - **SYMBOL**



- Byte 255 is special code marking
  - **EXCEPTION**

  followed by 1 uncompressed byte

Small symbol table(s):
RAM: 2.2KB,
disk/network: ~500B

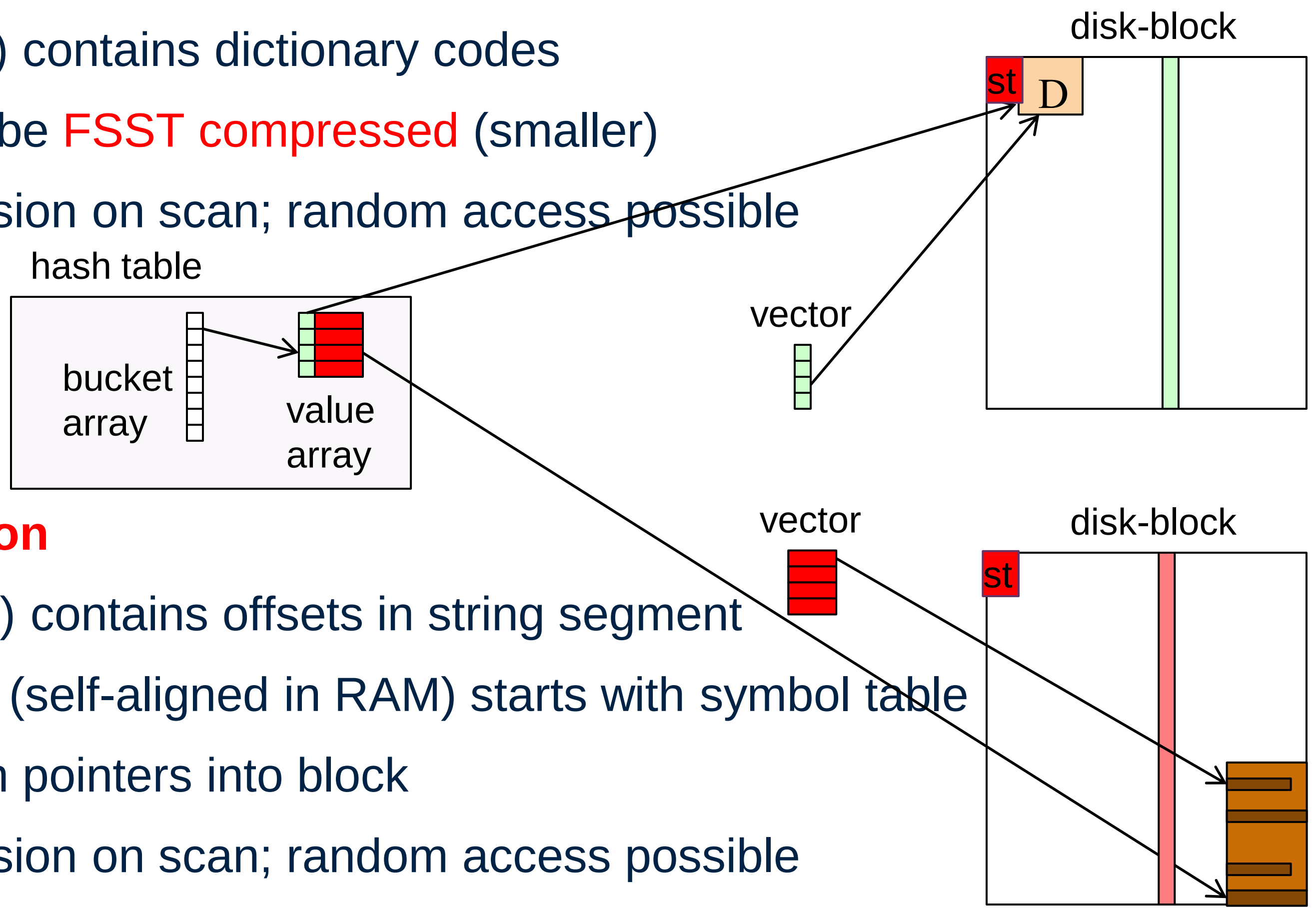
*Closest existing scheme is **RePair**, but is >100x slower than FSST (both ways)*

FSST bottom-up symbol table construction

- **Evolutionary-style algorithm**
- Starts with empty symbol table, uses 5 iterations:
  - We encode (a sample of) the plaintext with the current symbol table
    - We count the occurrence of each symbol
    - We count the occurrence of **each two subsequent** symbols
      - We also count single byte(-extension) frequencies, even if these are not symbols. For bootstrap and robustness.
  - Two subsequent symbols (or byte-extensions) generate a new **concatenated symbol**
  - We compute the **gain** (length*freq) of all bytes, old symbols and concatenated symbols and insert the 255 best in the new symbol table

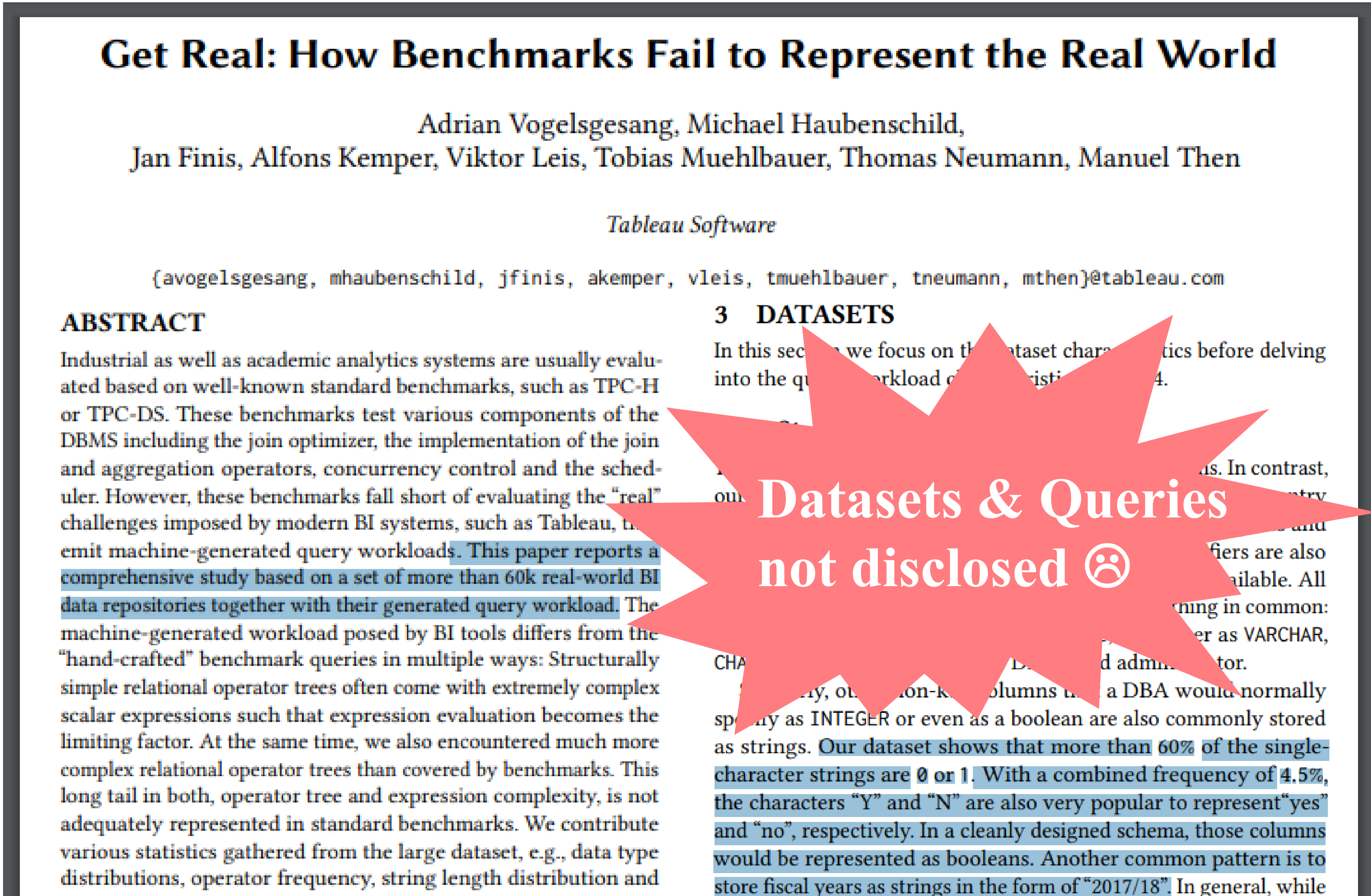
FSST Compression in a DBMS

- Dictionary Compression
  - column (green) contains dictionary codes
  - dictionary can be FSST compressed (smaller)
  - no decompression on scan; random access possible



- **FSST Compression**
  - column (brown) contains offsets in string segment
  - eg 64KB block (self-aligned in RAM) starts with symbol table
  - vectors contain pointers into block
  - no decompression on scan; random access possible

DBtest @ SIGMOD18



Get Real: How Benchmarks Fail to Represent the Real World

Adrian Vogelsgesang, Michael Haubenschild,
Jan Finis, Alfons Kemper, Viktor Leis, Tobias Muehlbauer, Thomas Neumann, Manuel Then

Tableau Software

{avogelsgesang, mhaubenschild, jfinis, akemper, vleis, tmuehlbauer, tneumann, mthen}@tableau.com

ABSTRACT

Industrial as well as academic analytics systems are usually evaluated based on well-known standard benchmarks, such as TPC-H or TPC-DS. These benchmarks test various components of the DBMS including the join optimizer, the implementation of the join and aggregation operators, concurrency control and the scheduler. However, these benchmarks fall short of evaluating the "real" challenges imposed by modern BI systems, such as Tableau, th... emit machine-generated query workloads. This paper reports a comprehensive study based on a set of more than 60k real-world BI data repositories together with their generated query workload. The machine-generated workload posed by BI tools differs from the "hand-crafted" benchmark queries in multiple ways: Structurally simple relational operator trees often come with extremely complex scalar expressions such that expression evaluation becomes the limiting factor. At the same time, we also encountered much more complex relational operator trees than covered by benchmarks. This long tail in both, operator tree and expression complexity, is not adequately represented in standard benchmarks. We contribute various statistics gathered from the large dataset, e.g., data type distributions, operator frequency, string length distribution and

3 DATASETS

In this sec... we focus on th... ataset chara... tics before delving into the qu... orkload c... risti... 4.

...s. In contrast, ou... ntry ... and ... fiers are also ... ailable. All ... ning in common: ... er as VARCHAR, CHA... D... d adm... tor.

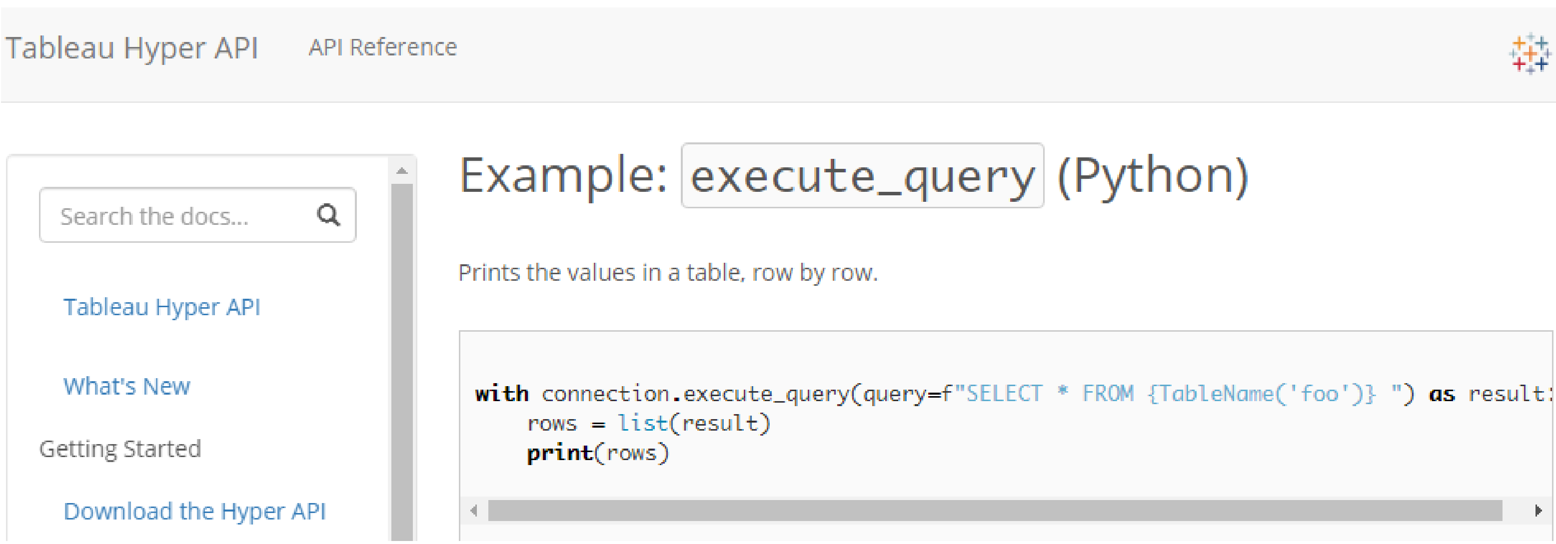
...y, ot... on-k... olumns th... a DBA would normally sp... ify as INTEGER or even as a boolean are also commonly stored as strings. Our dataset shows that more than 60% of the single-character strings are 0 or 1. With a combined frequency of 4.5%, the characters "Y" and "N" are also very popular to represent "yes" and "no", respectively. In a cleanly designed schema, those columns would be represented as booleans. Another common pattern is to store fiscal years as strings in the form of "2017/18". In general, while

Introducing: Public BI Benchmark

We downloaded the 50 biggest

Tableau Public Workbooks

- extracted data + (implicit) queries
- removed Tableau/Hyper-specific SQL



Get it from the CWI Database Architectures (DA) github:

github.com/cwida/public_bi_benchmark

Introducing: Public BI Benchmark

We downloaded the 50 biggest

Tableau Public Workbooks

- extracted data + (implicit) queries
- removed Tableau/Hyper-specific SQL

Workbook	Tables	Columns	Rows	Queries	CSV size
Arade	1	11	9.9M	1	811.4MiB
Bimbo	1	12	74.2M	2	3.0GiB
CMSprovider	2	52	18.6M	3	3.9GiB
CityMaxCapita	1	31	912.7K	10	333.0MiB
CommonGovernment	13	728	141.1M	38	102.5GiB
Corporations	1	27	741.7K	1	202.2MiB
Eixo	1	80	7.6M	24	6.4GiB
Euro2016	1	11	2.1M	1	390.6MiB
Food	1	6	5.2M	1	205.9MiB
Generico	5	215	114.1M	38	64.5GiB
HashTags	1	101	511.5K	12	640.2MiB
Hatred	1	31	873.2K	26	309.4MiB
IGlocations1	1	18	81.6K	3	6.6MiB
IGlocations2	2	40	4.3M	13	1.8GiB
IUBLibrary	1	27	1.8K	3	443.3KiB
MLB	68	3733	32.5M	95	8.2GiB
...					
Taxpayer	10	280	91.5M	22	17.1GiB
Telco	1	181	2.9M	1	2.3GiB
TrainsUK1	4	87	12.9M	8	3.9GiB
TrainsUK2	2	74	31.1M	1	12.2GiB
USCensus	3	1557	9.4M	8	13.6GiB
Uberlandia	1	81	7.6M	24	6.4GiB
Wins	4	2198	2.1M	13	3.9GiB
YaleLanguages	5	150	5.8M	13	1.5GiB
Total	206	13395	988.9M	646	386.5GiB

Table 3.1: Public BI benchmark workbooks

Get it from the CWI Database Architectures (DA) github:

github.com/cwida/public_bi_benchmark

A look at real BI user data

- **Dirty** Data (exceptions, errors)
- **Empty**/missing values that are not null (empty quotes, whitespace)

20 lines (20 sloc) | 15 KB

Raw | Blame | History

We can make this file beautiful and searchable if this error is corrected: It looks like row 9 should actually have 4 columns, instead of 2. in line 8.

```
 |-1|-1|0|    |1300229226|                                        |0|0|0|INDUSTRIAL PRODUCTS & SERVICES|TEST & MEASUREMENT SUPPLIES|423450|MEDICAL,
 |-1|-1|0|    |1300229226|                                        |0|0|0|INDUSTRIAL PRODUCTS & SERVICES|TEST & MEASUREMENT SUPPLIES|423450|MEDICAL,
-1|-1|0|    |1300229226|                                        |0|0|0|INDUSTRIAL PRODUCTS & SERVICES|TEST & MEASUREMENT SUPPLIES|423450|MEDICAL, DEN
 |-1|-1|0|    |1300229226|                                        |0|0|0|INDUSTRIAL PRODUCTS & SERVICES|TEST & MEASUREMENT SUPPLIES|423450|MEDICAL, D
-1|-1|0|    |1300229226|                                        |0|0|0|INDUSTRIAL PRODUCTS & SERVICES|TEST & MEASUREMENT SUPPLIES|423450|MEDICAL, DEN
-1|-1|0|    |1300229226|                                        |0|0|0|INDUSTRIAL PRODUCTS & SERVICES|TEST & MEASUREMENT SUPPLIES|423450|MEDICAL, DEN
 |-1|-1|0|    |1300229226|                                        |0|0|0|INDUSTRIAL PRODUCTS & SERVICES|TEST & MEASUREMENT SUPPLIES|423450|MEDICAL, D
-1|-1|0|    |1300229226|                                        |0|0|0|INDUSTRIAL PRODUCTS & SERVICES|TEST & MEASUREMENT SUPPLIES|423450|MEDICAL, DEN
     |-1|-1|0|null|1300229226|                                        |0|0|0|INDUSTRIAL PRODUCTS & SERVICES|TEST & MEASUREMENT SUPPLIES|339111|LABO
-1|-1|0|    |1300229226|                                        |0|0|0|MEDICAL|MEDICAL EQUIPMENT AND ACCESSORIES AND SUPPLIES|423450|MEDICAL, DENTAL,
```

A look at real BI user data

- **Dirty** Data (exceptions, errors)
- **Empty**/missing values that are not null (empty quotes, whitespace)
- Leading/trailing **whitespace** (fillers)

20 lines (20 sloc) | 15 KB

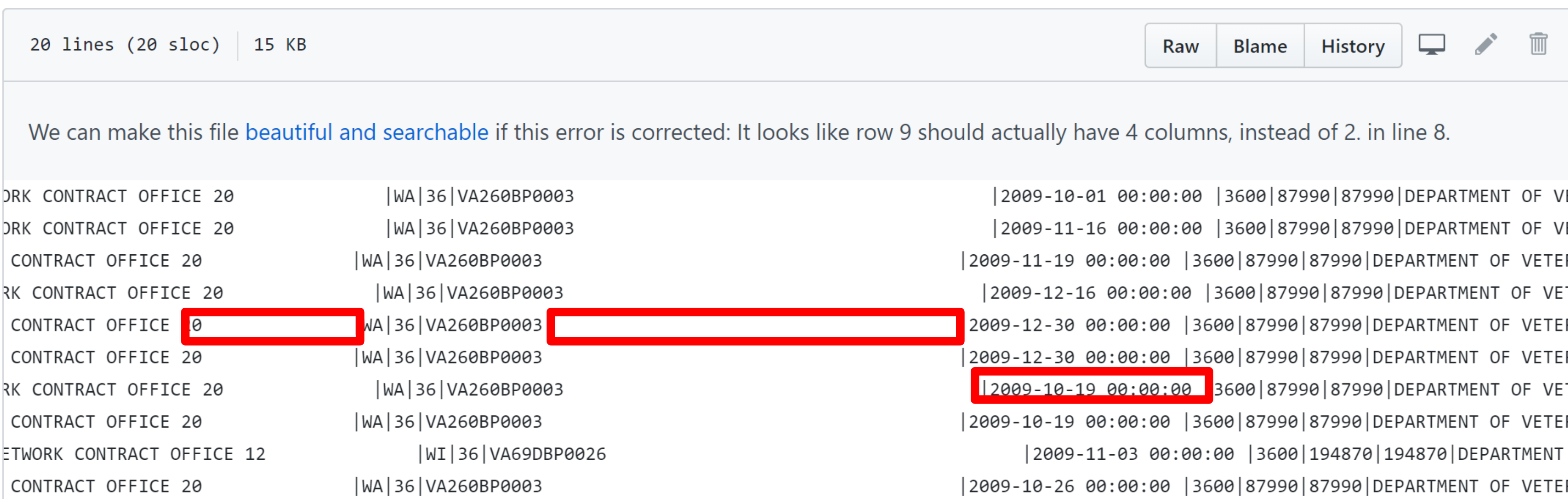
Raw | Blame | History

We can make this file beautiful and searchable if this error is corrected: It looks like row 9 should actually have 4 columns, instead of 2. in line 8.

```
ORK CONTRACT OFFICE 20              |WA|36|VA260BP0003                                                          |2009-10-01 00:00:00 |3600|87990|87990|DEPARTMENT OF VET
ORK CONTRACT OFFICE 20              |WA|36|VA260BP0003                                                          |2009-11-16 00:00:00 |3600|87990|87990|DEPARTMENT OF VET
 CONTRACT OFFICE 20              |WA|36|VA260BP0003                                                          |2009-11-19 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
RK CONTRACT OFFICE 20              |WA|36|VA260BP0003                                                          |2009-12-16 00:00:00 |3600|87990|87990|DEPARTMENT OF VETE
 CONTRACT OFFICE 0                WA|36|VA260BP0003                                                          2009-12-30 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
 CONTRACT OFFICE 20              |WA|36|VA260BP0003                                                          |2009-12-30 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
RK CONTRACT OFFICE 20              |WA|36|VA260BP0003                                                          |2009-10-19 00:00:00 |3600|87990|87990|DEPARTMENT OF VETE
 CONTRACT OFFICE 20              |WA|36|VA260BP0003                                                          |2009-10-19 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
ETWORK CONTRACT OFFICE 12              |WI|36|VA69DBP0026                                                          |2009-11-03 00:00:00 |3600|194870|194870|DEPARTMENT O
 CONTRACT OFFICE 20              |WA|36|VA260BP0003                                                          |2009-10-26 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
```

A look at real BI user data

- **Dirty** Data (exceptions, errors)
- **Empty**/missing values that are not null (empty quotes, whitespace)
- Leading/trailing **whitespace** (fillers)
- **Wrong typed**: eg numbers and dates stored in VARCHAR columns



20 lines (20 sloc) | 15 KB

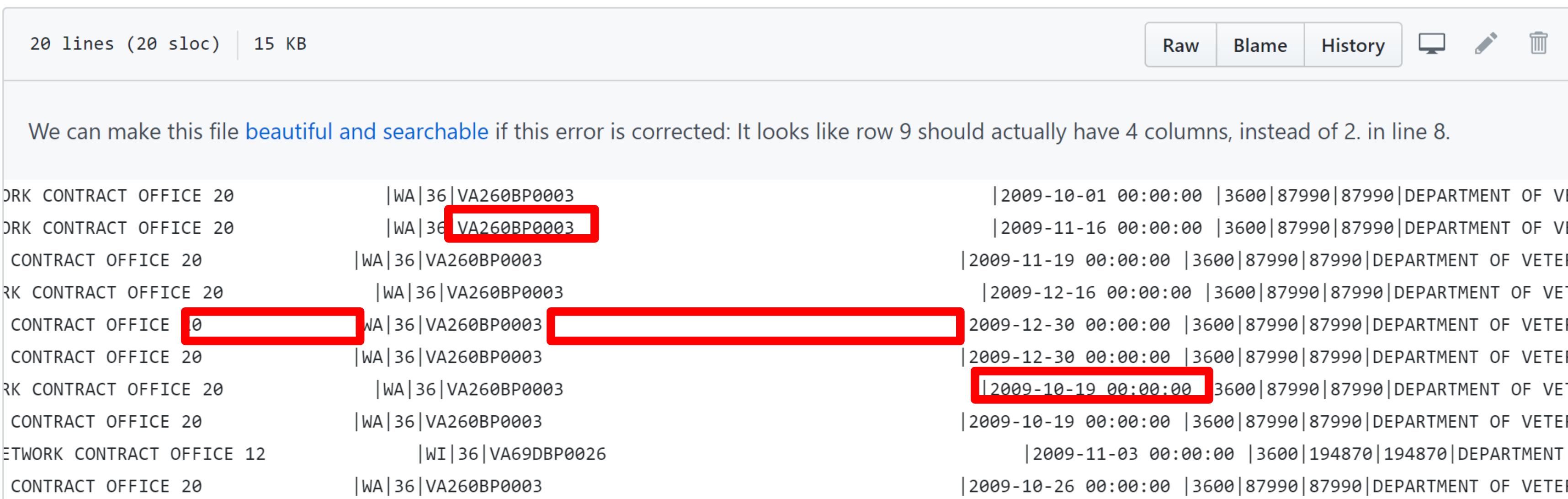
Raw | Blame | History

We can make this file beautiful and searchable if this error is corrected: It looks like row 9 should actually have 4 columns, instead of 2. in line 8.

```
ORK CONTRACT OFFICE 20          |WA|36|VA260BP0003                                        |2009-10-01 00:00:00 |3600|87990|87990|DEPARTMENT OF VET
ORK CONTRACT OFFICE 20          |WA|36|VA260BP0003                                        |2009-11-16 00:00:00 |3600|87990|87990|DEPARTMENT OF VET
 CONTRACT OFFICE 20          |WA|36|VA260BP0003                                        |2009-11-19 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
RK CONTRACT OFFICE 20          |WA|36|VA260BP0003                                        |2009-12-16 00:00:00 |3600|87990|87990|DEPARTMENT OF VETE
 CONTRACT OFFICE 0           |WA|36|VA260BP0003                                        |2009-12-30 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
 CONTRACT OFFICE 20          |WA|36|VA260BP0003                                        |2009-12-30 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
RK CONTRACT OFFICE 20          |WA|36|VA260BP0003                                        |2009-10-19 00:00:00 |3600|87990|87990|DEPARTMENT OF VETE
 CONTRACT OFFICE 20          |WA|36|VA260BP0003                                        |2009-10-19 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
ETWORK CONTRACT OFFICE 12          |WI|36|VA69DBP0026                                        |2009-11-03 00:00:00 |3600|194870|194870|DEPARTMENT C
 CONTRACT OFFICE 20          |WA|36|VA260BP0003                                        |2009-10-26 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
```

A look at real BI user data

- **Dirty** Data (exceptions, errors)
- **Empty**/missing values that are not null (empty quotes, whitespace)
- Leading/trailing **whitespace** (fillers)
- **Wrong typed**: eg numbers and dates stored in VARCHAR columns
- **Composed strings** from different types+distributions (eg emails, urls)


20 lines (20 sloc) | 15 KB

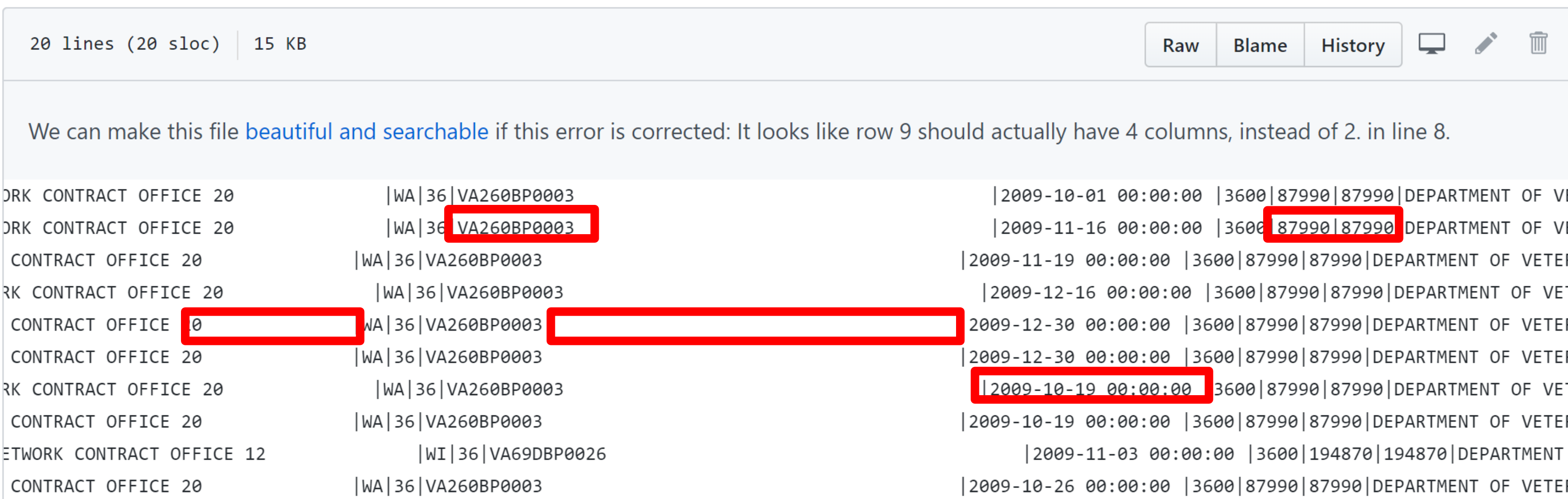
Raw Blame History

We can make this file beautiful and searchable if this error is corrected: It looks like row 9 should actually have 4 columns, instead of 2. in line 8.

```
ORK CONTRACT OFFICE 20          |WA|36|VA260BP0003                                      |2009-10-01 00:00:00 |3600|87990|87990|DEPARTMENT OF VET
ORK CONTRACT OFFICE 20          |WA|36 VA260BP0003                                      |2009-11-16 00:00:00 |3600|87990|87990|DEPARTMENT OF VET
 CONTRACT OFFICE 20          |WA|36|VA260BP0003                                      |2009-11-19 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
RK CONTRACT OFFICE 20          |WA|36|VA260BP0003                                      |2009-12-16 00:00:00 |3600|87990|87990|DEPARTMENT OF VETE
 CONTRACT OFFICE  0               WA|36|VA260BP0003                                       2009-12-30 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
 CONTRACT OFFICE 20          |WA|36|VA260BP0003                                      |2009-12-30 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
RK CONTRACT OFFICE 20          |WA|36|VA260BP0003                                      |2009-10-19 00:00:00 3600|87990|87990|DEPARTMENT OF VETE
 CONTRACT OFFICE 20          |WA|36|VA260BP0003                                      |2009-10-19 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
ETWORK CONTRACT OFFICE 12          |WI|36|VA69DBP0026                                      |2009-11-03 00:00:00 |3600|194870|194870|DEPARTMENT C
 CONTRACT OFFICE 20          |WA|36|VA260BP0003                                      |2009-10-26 00:00:00 |3600|87990|87990|DEPARTMENT OF VETERA
```

A look at real BI user data

- **Dirty** Data (exceptions, errors)
- **Empty**/missing values that are not null (empty quotes, whitespace)
- Leading/trailing **whitespace** (fillers)
- **Wrong typed**: eg numbers and dates stored in VARCHAR columns
- **Composed strings** from different types+distributions (eg emails, urls)
- **Correlations** between columns, or even repeated columns



Suboptimal Data Representations

- Negative effects
  - Data is much **larger** than needs to be
    - verbose strings, correlation=repetition, prevented dictionary compression
  - Queries take **more time** than they would need
    - expensive string processing, expensive casts, no predicate push-down

- "Users are doing a bad job" ➔ "should fix their data and schema"
  - This is **not going to happen**! End-users not even interested.
  - Move to cloud ➔ less DBA attention

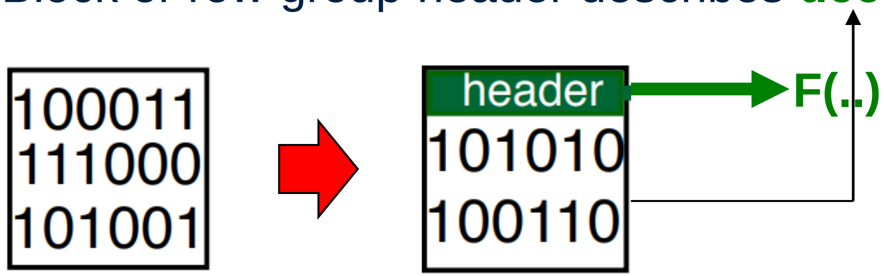
➔ systems should automatically compensate for suboptimal data

White-Box Compression one of the answers

- smaller data, more efficient query processing

White-Box Compression

- Configurable, data-dependent, compression schemes
  - Block or row-group header describes **decompression function**



Block of data

- Some **Research Questions** this raises:
  - What could these functions look like?
  - How does the system learn these functions during compression?
  - How much will compression rate improve?
  - How to exploit these functions in query optimization and execution?
  - How can a system quickly parse and execute such functions?

White-Box Compression Example

A	B
"GSA_8350"	"GENERAL SERVICES ADMINISTRATION"
"GSA_8351"	"GENERAL SERVICES ADMINISTRATION"
"HHS_2072"	"HEALTH AND HUMAN SERVICES"
"TREAS_4791"	"TREASURY"
"TREAS_4792"	"TREASURY"
"HHS_2073"	"HEALTH AND HUMAN SERVICES"
"GSA_8352"	"GENERAL SERVICES ADMINISTRATION"

CWI
Database Architectures

White-Box Compression Example

Logical		Physical	
A	B	P	Q
"GSA_8350"	"GENERAL SERVICES ADMINISTRATION"	0	8350
"GSA_8351"	"GENERAL SERVICES ADMINISTRATION"	0	8351
"HHS_2072"	"HEALTH AND HUMAN SERVICES"	1	2072
"TREAS_4791"	"TREASURY"	2	4791
"TREAS_4792"	"TREASURY"	2	4792
"HHS_2073"	"HEALTH AND HUMAN SERVICES"	1	2073
"GSA_8352"	"GENERAL SERVICES ADMINISTRATION"	0	8352

Less Storage +
Better Compression

$$A = concat(map(P, dict_{AP}), const(\texttt{"\_"}), format(Q, \texttt{"\%d"}))$$
$$B = map(P, dict_{BP})$$

key	value
0	"GSA"
1	"HHS"
2	"TREAS"

$dict_{AP}$

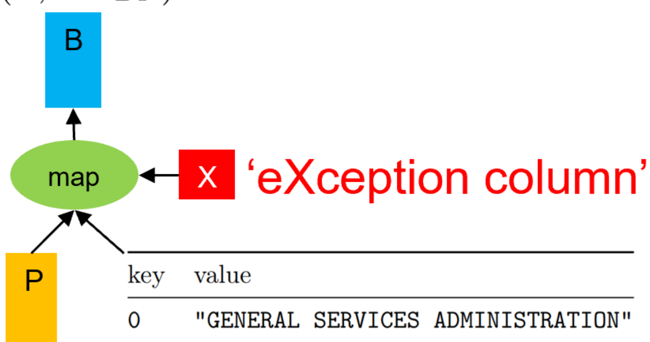
key	value
0	"GENERAL SERVICES ADMINISTRATION"
1	"HEALTH AND HUMAN SERVICES"
2	"TREASURY"

$dict_{BP}$

White-Box Compression Example

Logical		Physical	
A	B	P	Q
"GSA_8350"	"GENERAL SERVICES ADMINISTRATION"	0	8350
"GSA_8351"	"GENERAL SERVICES ADMINISTRATION"	0	8351
"HHS_2072"	"HEALTH AND HUMAN SERVICES"	1	2072
"TREAS_4791"	"TREASURY"	2	4791
"TREAS_4792"	"TREASURY"	2	4792
"HHS_2073"	"HEALTH AND HUMAN SERVICES"	1	2073
"GSA_8352"	"GENERAL SERVICES ADMINISTRATION"	0	8352

$$A = concat(map(P, dict_{AP}), const(\texttt{"\_"}), format(Q, \texttt{"\%d"}))$$
$$B = map(P, dict_{BP})$$



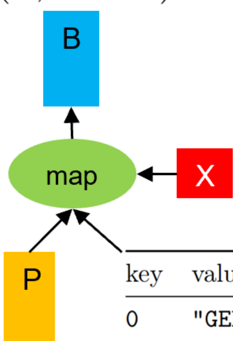
key	value
0	"GENERAL SERVICES ADMINISTRATION"
1	"HEALTH AND HUMAN SERVICES"
2	"TREASURY"

$dict_{BP}$

White-Box Compression Example

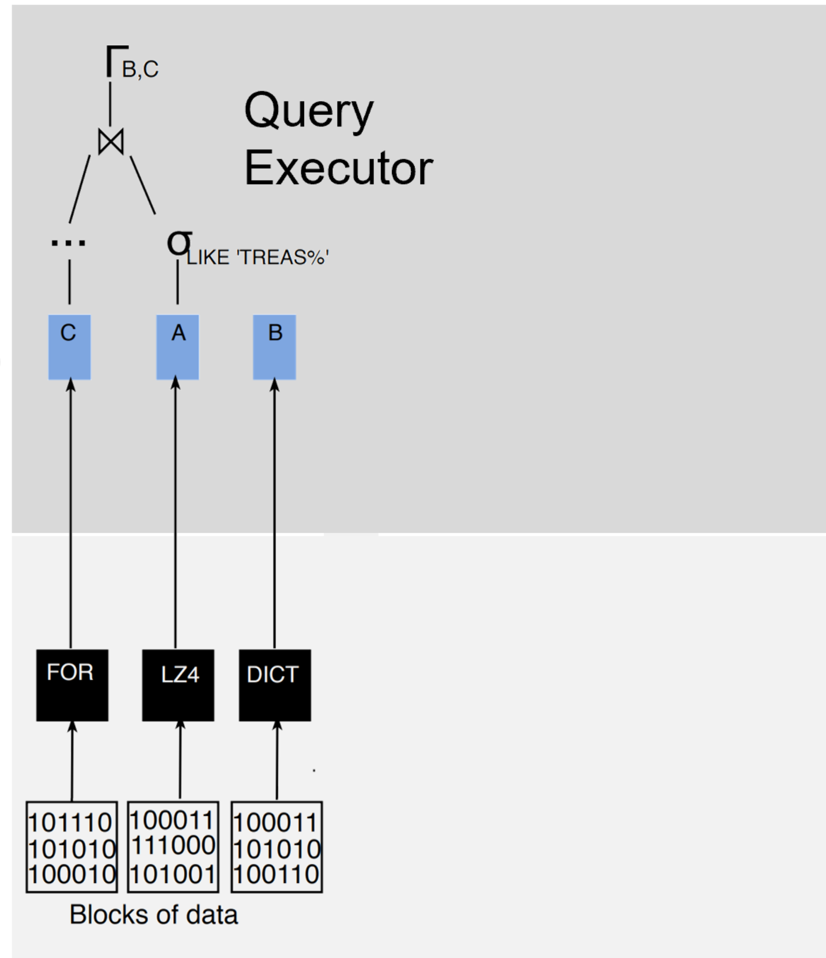
Logical		Physical	
A	B	P	Q
"GSA_8350"	"GENERAL SERVICES ADMINISTRATION"	0	8350
"GSA_8351"	"GENERAL SERVICES ADMINISTRATION"	0	8351
"HHS_2072"	"HEALTH AND HUMAN SERVICES"	1	2072
"TREAS_4791"	"TREASURY"	2	4791
"TREAS_4792"	"TREASURY"	2	4792
"HHS_2073"	"HEALTH AND HUMAN SERVICES"	1	2073
"GSA_8352"	"GENERAL SERVICES ADMINISTRATION"	0	8352

$$A = concat(map(P, dict_{AP}), const(\texttt{"\_"}), format(Q, \texttt{"\%d"}))$$
$$B = map(P, dict_{BP})$$



key	value
0	"GENERAL SERVICES ADMINISTRATION"
1	"HEALTH AND HUMAN SERVICES"
2	"TREASURY"

$dict_{BP}$




```
SELECT tab.B, dim.C
FROM   tab JOIN dim ON tab.B = dim.C
WHERE  tab.A LIKE 'TREAS%'
```

CWI
Database Architectures

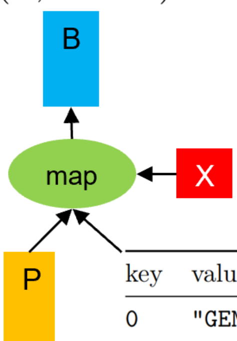
White-Box Compression Example

key	value
0	"GSA"
1	"HHS"
2	"TREAS"

$dict_{AP}$

Logical		Physical	
A	B	P	Q
"GSA_8350"	"GENERAL SERVICES ADMINISTRATION"	0	8350
"GSA_8351"	"GENERAL SERVICES ADMINISTRATION"	0	8351
"HHS_2072"	"HEALTH AND HUMAN SERVICES"	1	2072
"TREAS_4791"	"TREASURY"	2	4791
"TREAS_4792"	"TREASURY"	2	4792
"HHS_2073"	"HEALTH AND HUMAN SERVICES"	1	2073
"GSA_8352"	"GENERAL SERVICES ADMINISTRATION"	0	8352

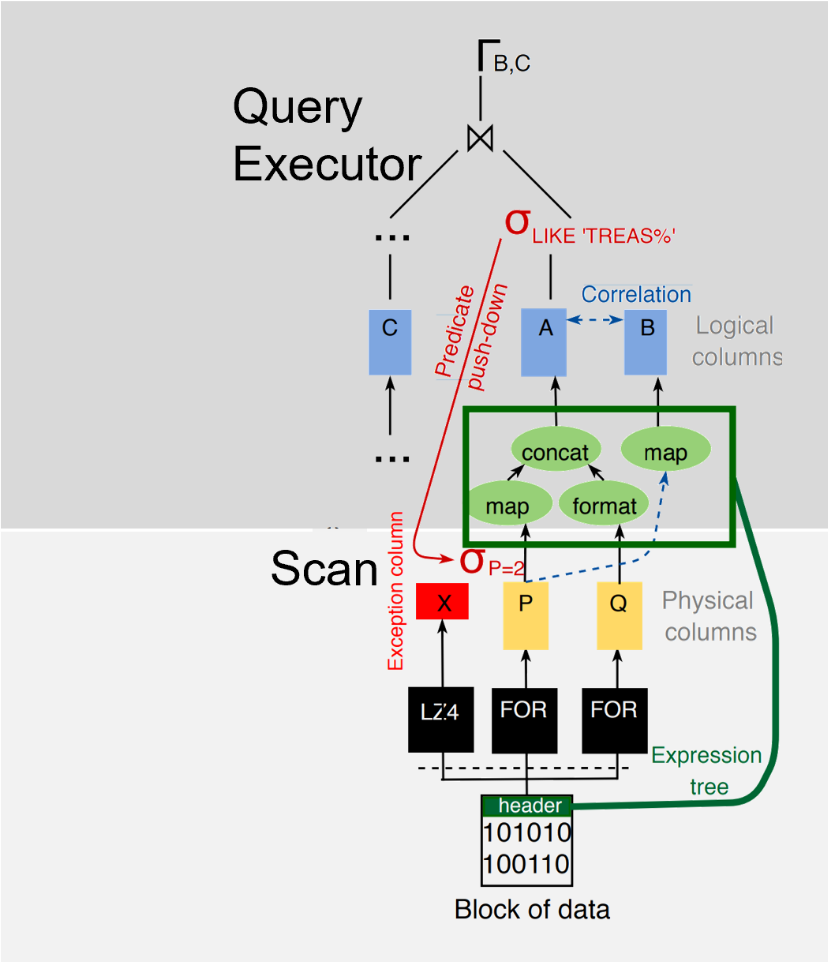
$$A = concat(map(P, dict_{AP}), const(\texttt{"\_"}), format(Q, \texttt{"\%d"}))$$
$$B = map(P, dict_{BP})$$



key	value
0	"GENERAL SERVICES ADMINISTRATION"
1	"HEALTH AND HUMAN SERVICES"
2	"TREASURY"

$dict_{BP}$

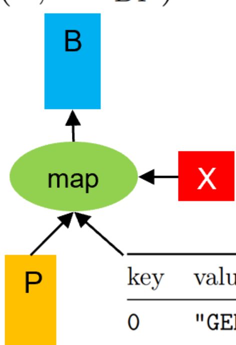
```
SELECT tab.B, dim.C
FROM   tab JOIN dim ON tab.B = dim.C
WHERE  tab.A LIKE 'TREAS%'
```



White-Box Compression Example

Logical		Physical	
A	B	P	Q
"GSA_8350"	"GENERAL SERVICES ADMINISTRATION"	0	8350
"GSA_8351"	"GENERAL SERVICES ADMINISTRATION"	0	8351
"HHS_2072"	"HEALTH AND HUMAN SERVICES"	1	2072
"TREAS_4791"	"TREASURY"	2	4791
"TREAS_4792"	"TREASURY"	2	4792
"HHS_2073"	"HEALTH AND HUMAN SERVICES"	1	2073
"GSA_8352"	"GENERAL SERVICES ADMINISTRATION"	0	8352

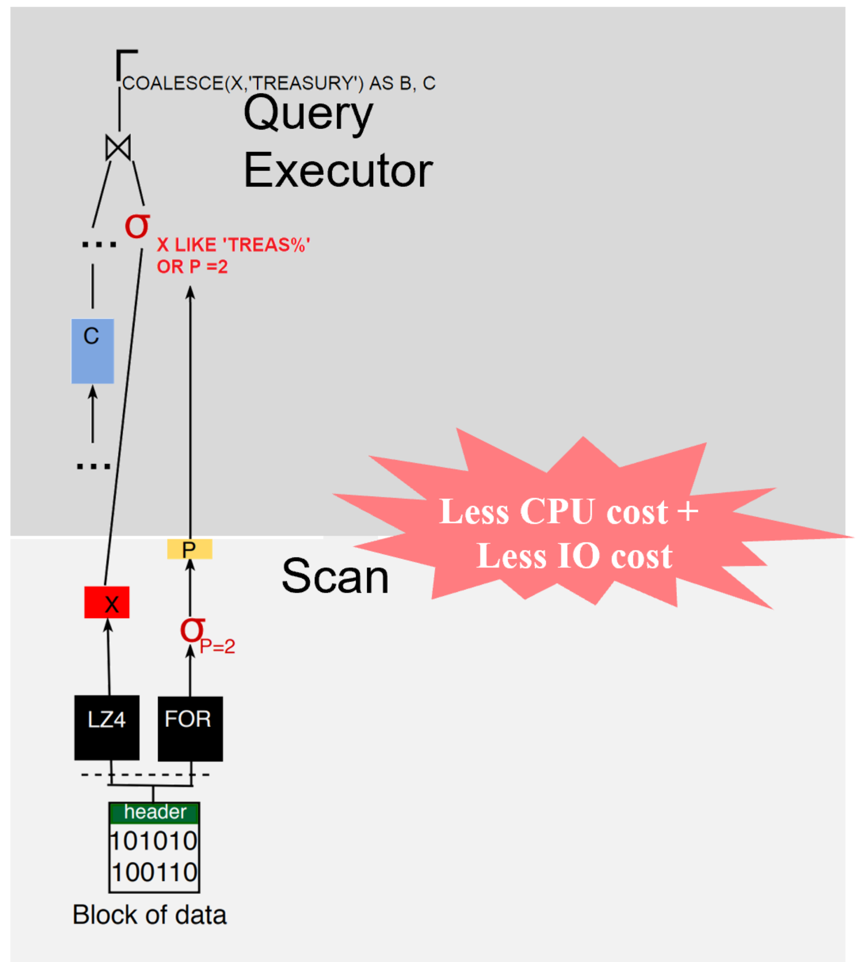
$$A = concat(map(P, dict_{AP}), const(\texttt{"\_"}), format(Q, \texttt{"\%d"}))$$
$$B = map(P, dict_{BP})$$



key	value
0	"GENERAL SERVICES ADMINISTRATION"
1	"HEALTH AND HUMAN SERVICES"
2	"TREASURY"

$dict_{BP}$

```
SELECT tab.B, dim.C
FROM   tab JOIN dim ON tab.B = dim.C
WHERE  tab.A LIKE 'TREAS%'
```



Summary

- **USSR string compression:** global delta-compression on-the fly
  - Transforms string operations into integer operations
  - Smaller & faster hash tables (joins, aggregates) – “optimistic splitting”
- **FSST string compression:** makes strings ~2x shorter
  - Allows random-access ➔ predicate pushdown + compressed execution
  - Faster decompression and better ratios than LZ4 + snappy!!
  - MIT licensed, code, paper + replication package github.com/cwida/fsst
  - System-architectures challenge: managing multiple symbol tables in-flight
- **White-box compression** learns better table representations

  BI users create poorly shaped datasets, likely won’t change
  - smaller storage (better datatypes, less redundancy)
  - compression expressions are learned from the data!
  - Lot’s of angles of research here (it is a learning problem!)

Public BI Benchmark github.com/cwida/public_bi_benchmark