Intro to Database Systems (15-445/645)

05 Storage Models & Compression

Carnegie Mellon University

FALL 2022

Andy Pavlo

ADMINISTRIVIA

Homework #2 is due September 25th @ 11:59pm

Project #1 is due October 2nd @ 11:59pm

DATABASE WORKLOADS

On-Line Transaction Processing (OLTP)

→ Fast operations that only read/update a small amount of data each time.

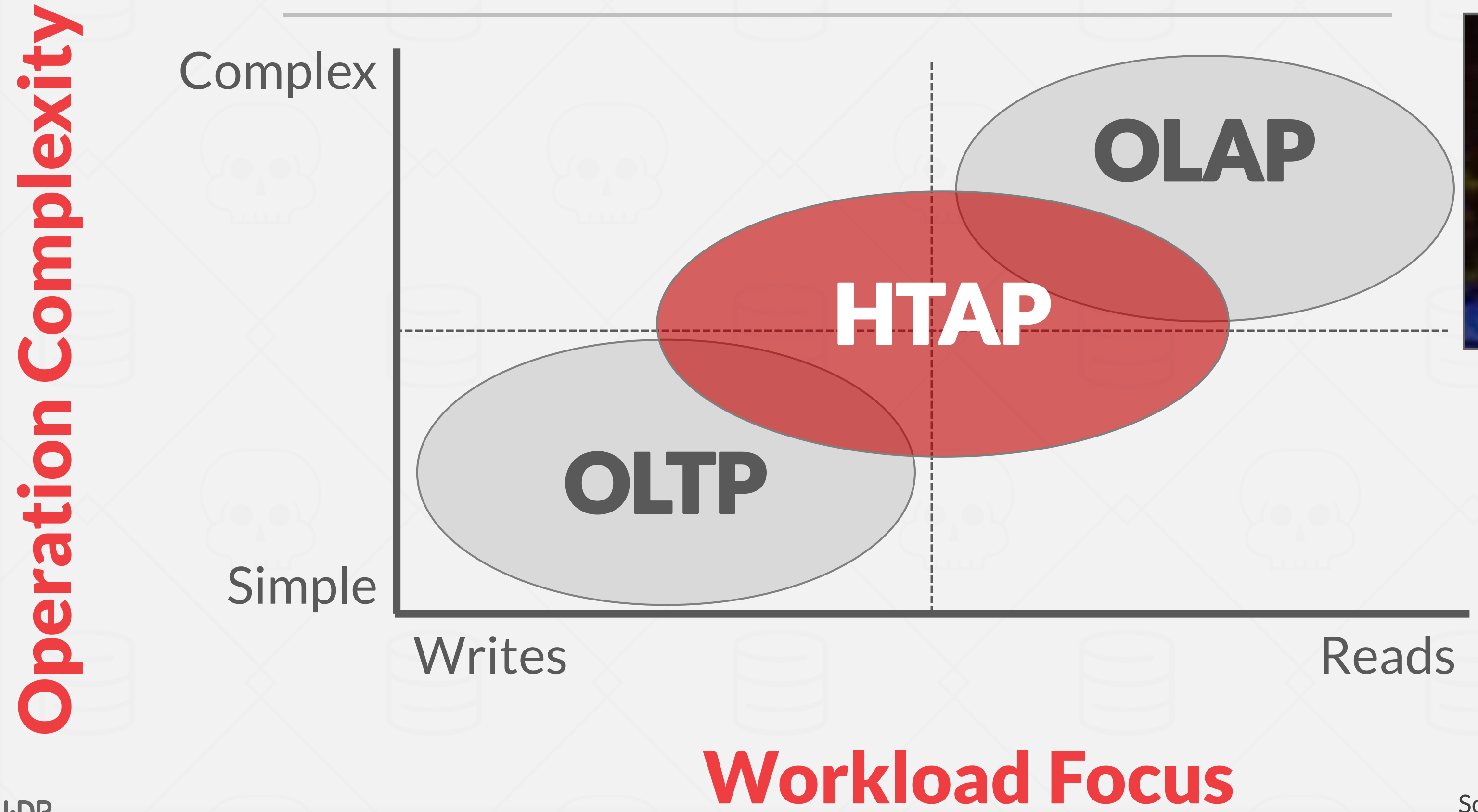
On-Line Analytical Processing (OLAP)

→ Complex queries that read a lot of data to compute aggregates.

Hybrid Transaction + Analytical Processing

→ OLTP + OLAP together on the same database instance

DATABASE WORKLOADS



Jim Gray

Source: Mike Stonebraker

WIKIPEDIA EXAMPLE

```
CREATE TABLE useracct (
  userID INT PRIMARY KEY,
  userName VARCHAR UNIQUE,
  ⋮
);
```

```
CREATE TABLE pages (
  pageID INT PRIMARY KEY,
  title VARCHAR UNIQUE,
  latest INT
  ↳REFERENCES revisions (revID),
);
```

```
CREATE TABLE revisions (
  revID INT PRIMARY KEY,
  userID INT REFERENCES useracct (userID),
  pageID INT REFERENCES pages (pageID),
  content TEXT,
  updated DATETIME
);
```

OBSERVATION

The relational model does not specify that the DBMS must store all a tuple's attributes together in a single page.

This may not actually be the best layout for some workloads…

OLTP

On-line Transaction Processing:

- → Simple queries that read/update a small amount of data that is related to a single entity in the database.

This is usually the kind of application that people build first.

```
SELECT P.*, R.*
  FROM pages AS P
 INNER JOIN revisions AS R
    ON P.latest = R.revID
 WHERE P.pageID = ?
```

```
UPDATE useracct
   SET lastLogin = NOW(),
       hostname = ?
 WHERE userID = ?
```

```
INSERT INTO revisions VALUES
(?,?…,?)
```

OLAP

On-line Analytical Processing:

→ Complex queries that read large portions of the database spanning multiple entities.

You execute these workloads on the data you have collected from your OLTP application(s).

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM
            U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY
  EXTRACT(month FROM U.lastLogin)
```

DATA STORAGE MODELS

The DBMS can store tuples in different ways that are better for either OLTP or OLAP workloads.

We have been assuming the **n-ary storage model** (aka "row storage") so far this semester.

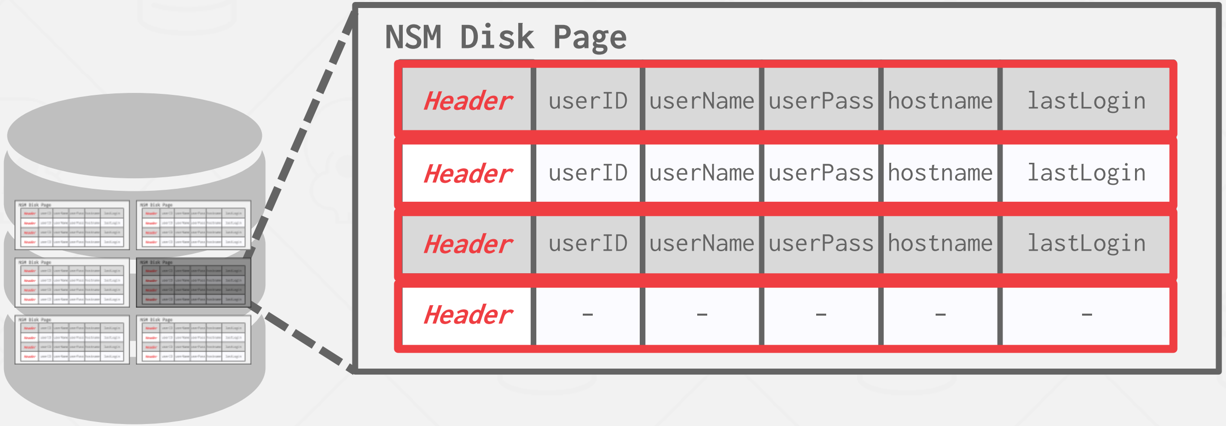
N-ARY STORAGE MODEL (NSM)

The DBMS stores all attributes for a single tuple contiguously in a page.

Ideal for OLTP workloads where queries tend to operate only on an individual entity and insert-heavy workloads.

N-ARY STORAGE MODEL (NSM)

The DBMS stores all attributes for a single tuple contiguously in a page.

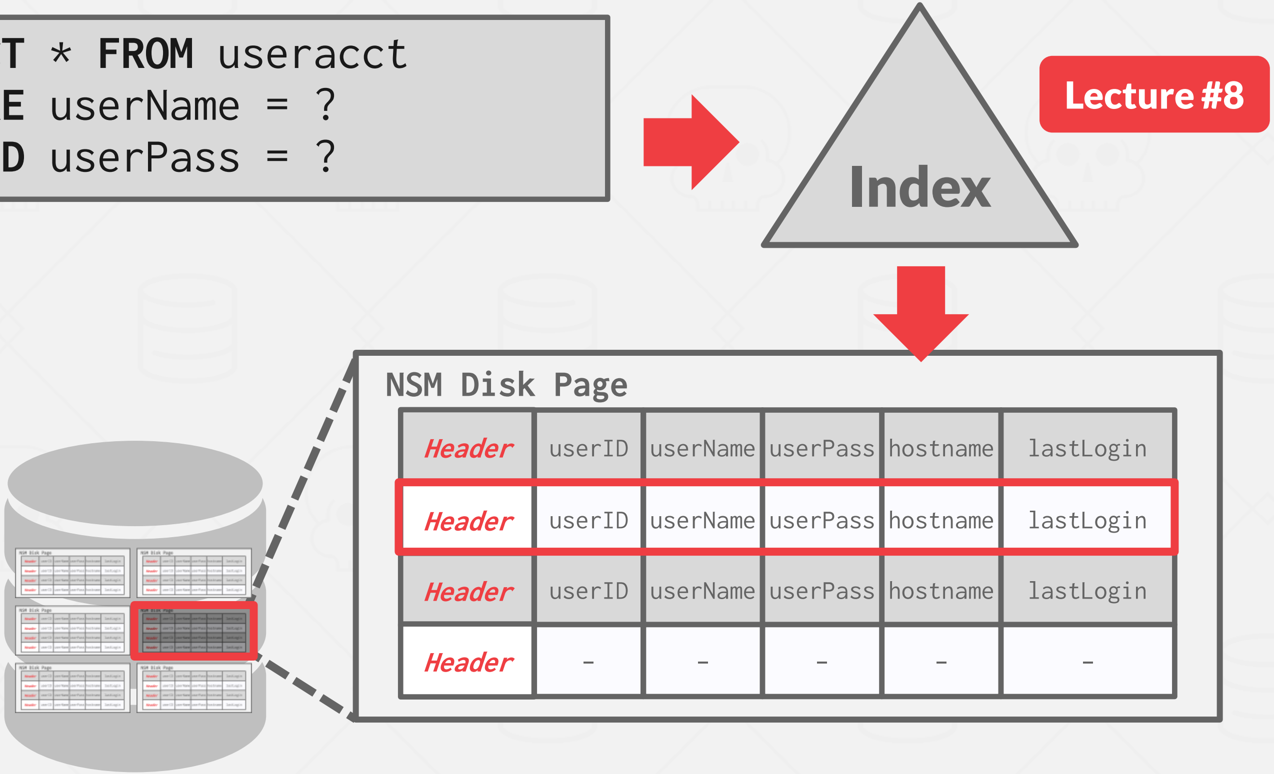


NSM Disk Page

Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	-	-	-	-	-

N-ARY STORAGE MODEL (NSM)

```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```



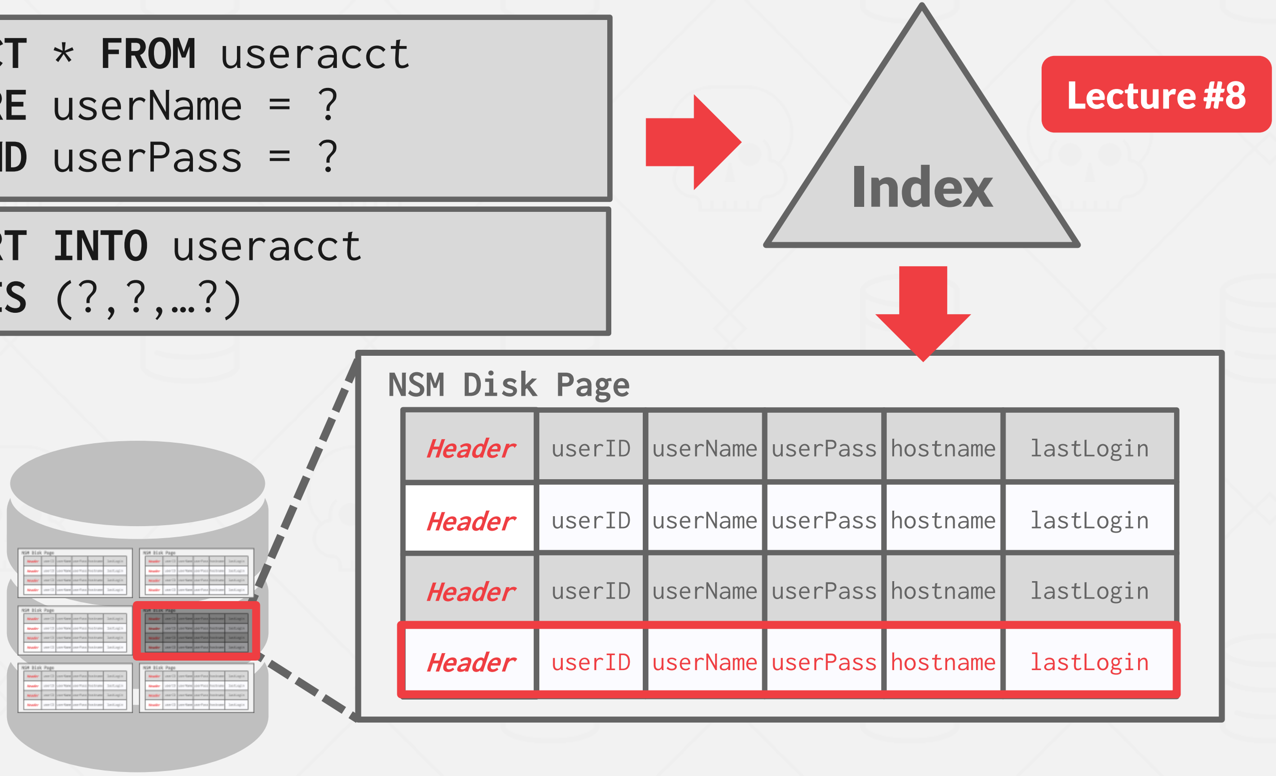
NSM Disk Page

Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	-	-	-	-	-

N-ARY STORAGE MODEL (NSM)

```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```

```
INSERT INTO useracct
VALUES (?,?,…?)
```

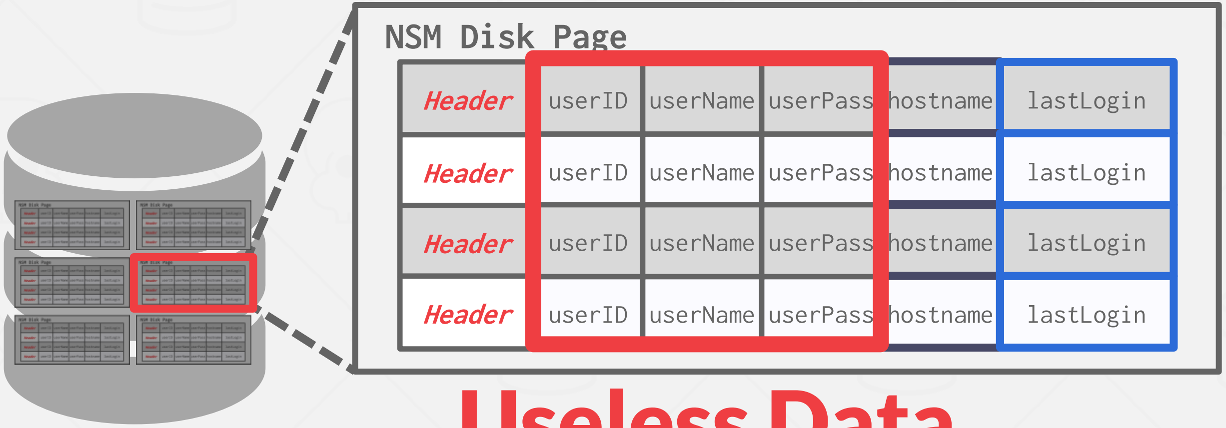


NSM Disk Page

Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

N-ARY STORAGE MODEL (NSM)

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



NSM Disk Page

Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

Useless Data

N-ARY STORAGE MODEL

Advantages

→ Fast inserts, updates, and deletes.
→ Good for queries that need the entire tuple.

Disadvantages

→ Not good for scanning large portions of the table and/or a subset of the attributes.

DECOMPOSITION STORAGE MODEL (DSM)

The DBMS stores the values of a single attribute for all tuples contiguously in a page.

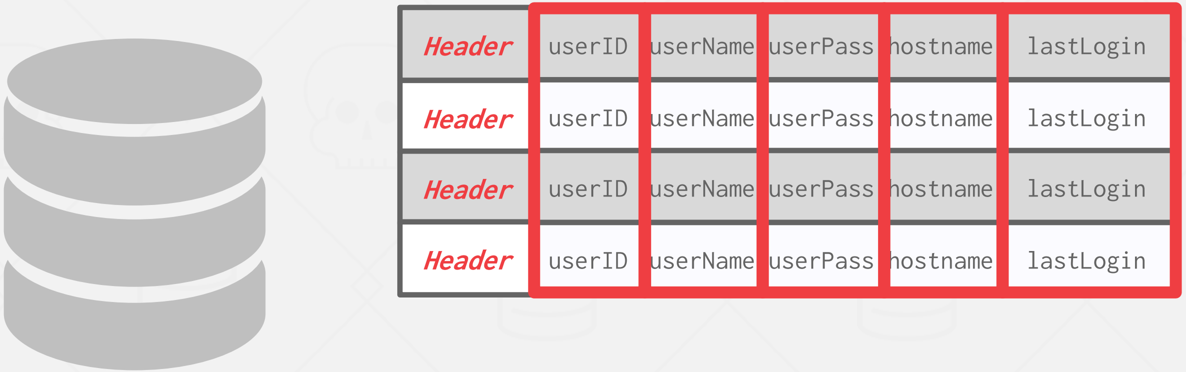
- → Also known as a "column store"

Ideal for OLAP workloads where read-only queries perform large scans over a subset of the table's attributes.

DECOMPOSITION STORAGE MODEL (DSM)

The DBMS stores the values of a single attribute across multiple tuples contiguously in a page.

→ Also known as a "column store".

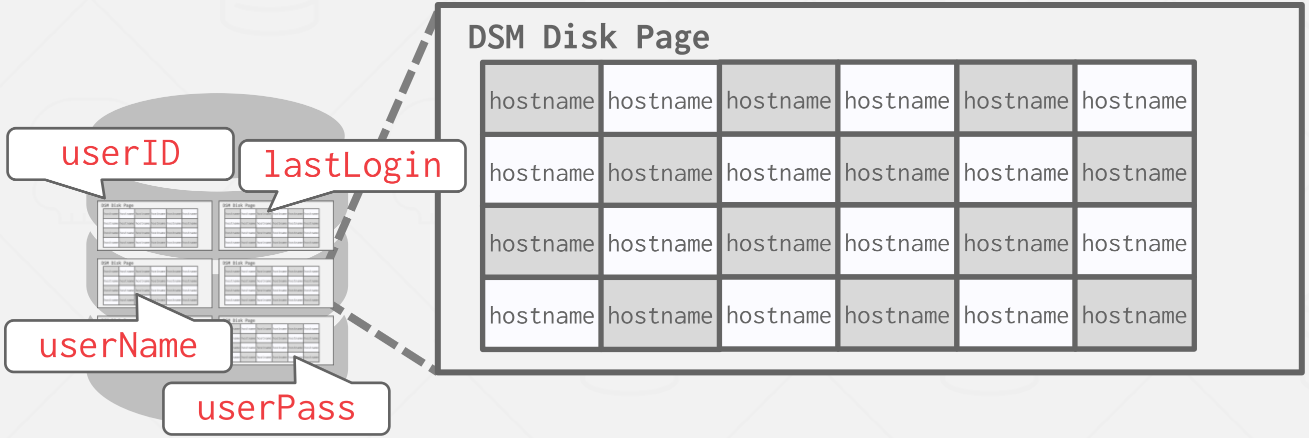


Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

DECOMPOSITION STORAGE MODEL (DSM)

The DBMS stores the values of a single attribute across multiple tuples contiguously in a page.

→ Also known as a "column store".

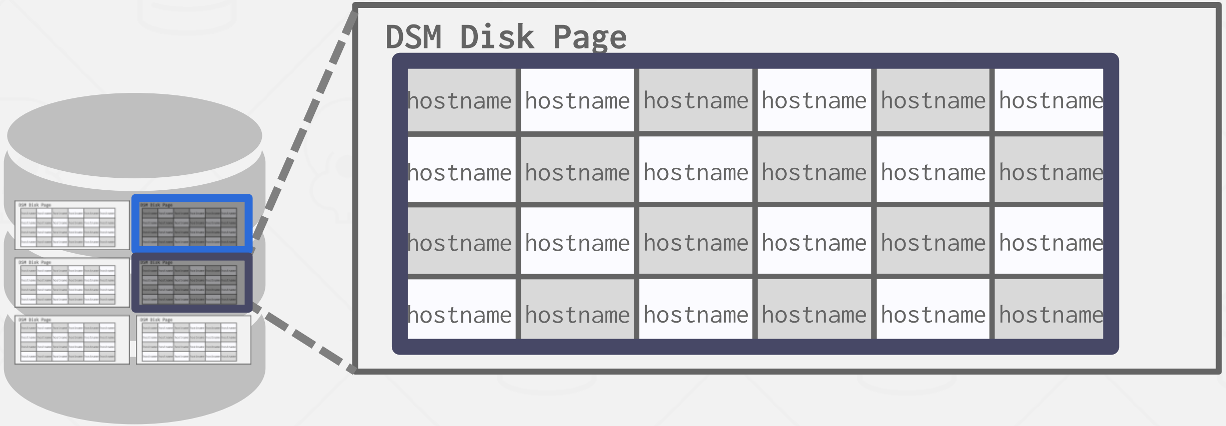


DSM Disk Page

hostname	hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname	hostname

DECOMPOSITION STORAGE MODEL (DSM)

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



DSM Disk Page

hostname	hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname	hostname
hostname	hostname	hostname	hostname	hostname	hostname

TUPLE IDENTIFICATION

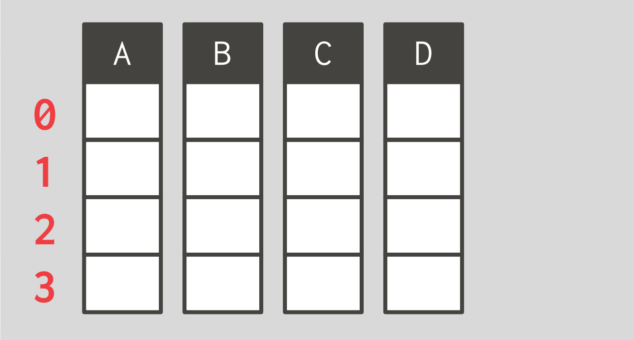
Choice #1: Fixed-length Offsets

→ Each value is the same length for an attribute.

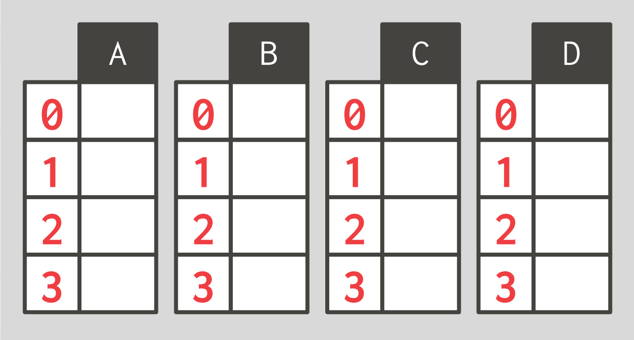
Choice #2: Embedded Tuple Ids

→ Each value is stored with its tuple id in a column.

Offsets



Embedded Ids



DECOMPOSITION STORAGE MODEL (DSM)

Advantages

- → Reduces the amount wasted I/O because the DBMS only reads the data that it needs.
- → Better query processing and data compression (more on this later).

Disadvantages

- → Slow for point queries, inserts, updates, and deletes because of tuple splitting/stitching.

DSM SYSTEM HISTORY

1970s: Cantor DBMS

1980s: DSM Proposal

1990s: SybaseIQ (in-memory only)

2000s: Vertica, VectorWise, MonetDB

2010s: Everyone

SYBASE
VERTICA
monetdb
vectorwise

pinot
presto
SAP HANA
ClickHouse
UMBRA
QuestDB
ORACLE
druid
cloudera IMPALA
amazon REDSHIFT
trino
Yellowbrick
HIVE
ARESDB
Microsoft SQL Server
Exasol
Greenplum
HYRISE
MariaDB
DuckDB
noise page
Yellowbrick
IBM DB2
SingleStore
InfiniDB
FIREBOLT

OBSERVATION

I/O is the main bottleneck if the DBMS fetches data from disk during query execution.

The DBMS can **compress** pages to increase the utility of the data moved per I/O operation.

Key trade-off is speed vs. compression ratio
→ Compressing the database reduces DRAM requirements.
→ It may decrease CPU costs during query execution.

REAL-WORLD DATA CHARACTERISTICS

Data sets tend to have highly skewed distributions for attribute values.

→ Example: Zipfian distribution of the Brown Corpus

Data sets tend to have high correlation between attributes of the same tuple.

→ Example: Zip Code to City, Order Date to Ship Date

DATABASE COMPRESSION

Goal #1: Must produce fixed-length values.
- → Only exception is var-length data stored in separate pool.

Goal #2: Postpone decompression for as long as possible during query execution.
- → Also known as late materialization.

Goal #3: Must be a lossless scheme.

LOSSLESS VS. LOSSY COMPRESSION

When a DBMS uses compression, it is always **lossless** because people don't like losing data.

Any kind of **lossy** compression must be performed at the application level.

COMPRESSION GRANULARITY

Choice #1: Block-level
- → Compress a block of tuples for the same table.

Choice #2: Tuple-level
- → Compress the contents of the entire tuple (NSM-only).

Choice #3: Attribute-level
- → Compress a single attribute within one tuple (overflow).
- → Can target multiple attributes for the same tuple.

Choice #4: Column-level
- → Compress multiple values for one or more attributes stored for multiple tuples (DSM-only).

NAÏVE COMPRESSION

Compress data using a general-purpose algorithm. Scope of compression is only based on the data provided as input.

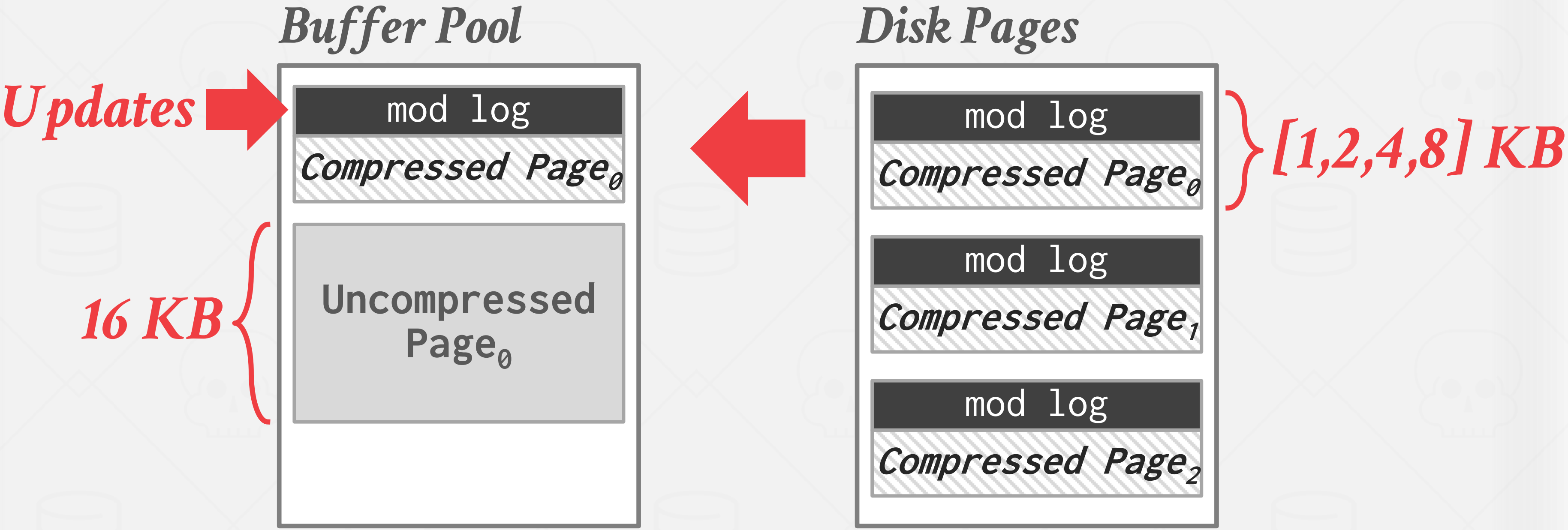
→ LZO (1996), LZ4 (2011), Snappy (2011), Oracle OZIP (2014), Zstd (2015)

Considerations

→ Computational overhead

→ Compress vs. decompress speed.

MYSQL INNODB COMPRESSION



Source: MySQL 5.7 Documentation

NAÏVE COMPRESSION

The DBMS must decompress data first before it can be read and (potentially) modified.

→ This limits the "scope" of the compression scheme.

These schemes also do not consider the high-level meaning or semantics of the data.

OBSERVATION

Ideally, we want the DBMS to operate on compressed data without decompressing it first.

Database Magic!

```
SELECT * FROM users
 WHERE name = 'Andy'
```

```
SELECT * FROM users
 WHERE name = XX
```

NAME	SALARY
Andy	99999
Matt	88888

NAME	SALARY
XX	AA
YY	BB

COMPRESSION GRANULARITY

Choice #1: Block-level

→ Compress a block of tuples for the same table.

Choice #2: Tuple-level

→ Compress the contents of the entire tuple (NSM-only).

Choice #3: Attribute-level

→ Compress a single attribute within one tuple (overflow).

→ Can target multiple attributes for the same tuple.

Choice #4: Column-level

→ Compress multiple values for one or more attributes stored for multiple tuples (DSM-only).

COLUMNAR COMPRESSION

Run-length Encoding
Bit-Packing Encoding
Bitmap Encoding
Delta Encoding
Incremental Encoding
Dictionary Encoding

RUN-LENGTH ENCODING

Compress runs of the same value in a single column into triplets:

- → The value of the attribute.
- → The start position in the column segment.
- → The # of elements in the run.

Requires the columns to be sorted intelligently to maximize compression opportunities.

RUN-LENGTH ENCODING

Original Data

id	sex
1	M
2	M
3	M
4	F
6	M
7	F
8	M
9	M

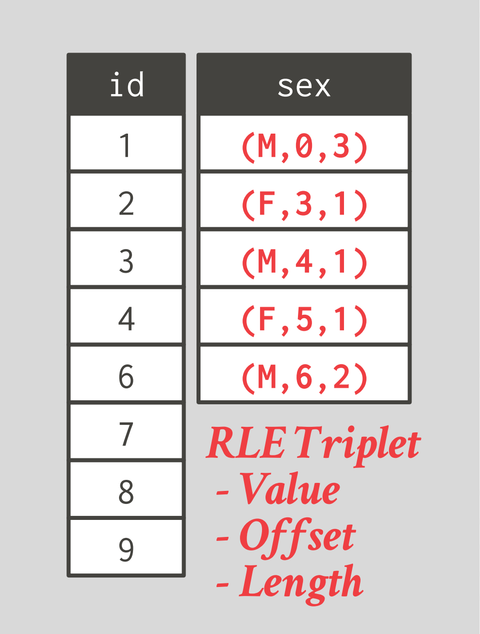
Compressed Data

id	sex
1	(M,0,3)
2	(F,3,1)
3	(M,4,1)
4	(F,5,1)
6	(M,6,2)
7	
8	
9	

RLE Triplet
- *Value*
- *Offset*
- *Length*

RUN-LENGTH ENCODING

```
SELECT sex, COUNT(*)
  FROM users
 GROUP BY sex
```



Compressed Data

id	sex
1	(M,0,3)
2	(F,3,1)
3	(M,4,1)
4	(F,5,1)
6	(M,6,2)
7	
8	
9	

RLE Triplet
- *Value*
- *Offset*
- *Length*

RUN-LENGTH ENCODING

Original Data

id	sex
1	M
2	M
3	M
4	F
6	M
7	F
8	M
9	M

Compressed Data

id	sex
1	(M,0,3)
2	(F,3,1)
3	(M,4,1)
4	(F,5,1)
6	(M,6,2)
7	
8	
9	

RLE Triplet
- *Value*
- *Offset*
- *Length*

RUN-LENGTH ENCODING

Sorted Data

id	sex
1	M
2	M
3	M
6	M
8	M
9	M
4	F
7	F

Compressed Data

id	sex
1	(M,0,6)
2	(F,7,2)
3	
6	
8	
9	
4	
7	

BIT-PACKING ENCODING

When values for an attribute are always less than the value's declared largest size, store them as smaller data type.

Original Data

int64
2
4
45
6
18

BIT-PACKING ENCODING

When values for an attribute are always less than the value's declared largest size, store them as smaller data type.

Original Data

Compressed Data

5 × 64-bits = 320 bits

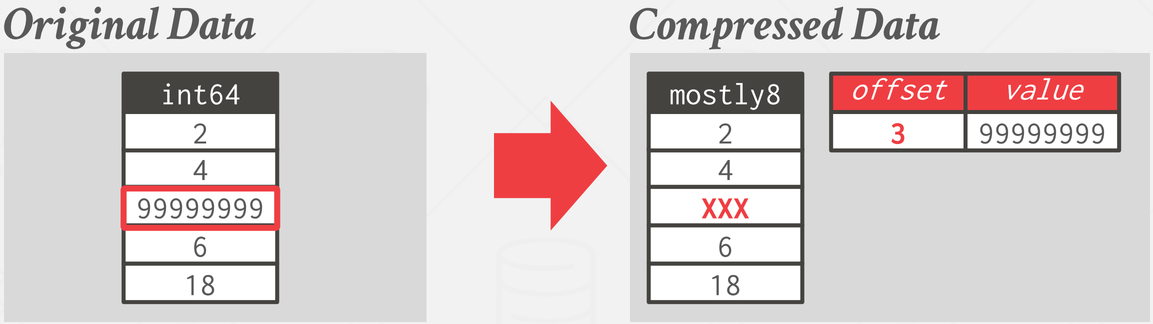
int64
00000000000000000000000000000000 00000000000000000000000000000010
00000000000000000000000000000000 00000000000000000000000000000011
00000000000000000000000000000000 00000000000000000000000000011101
00000000000000000000000000000000 00000000000000000000000000000110
00000000000000000000000000000000 00000000000000000000000000010010

packed-int8
00000010
00000011
00011101
00000110
00010010

(5 × 8-bits) = 40 bits

MOSTLY ENCODING

Bit-packing variant that uses a special marker to indicate when a value exceeds largest size and then maintain a look-up table to store them.



Original Data

int64
2
4
99999999
6
18

5 × 64-bits = 320 bits

Compressed Data

mostly8
2
4
XXX
6
18

offset	*value*
3	99999999

(5 × 8-bits) + 16-bits + 64-bits = 120 bits

Source: Redshift Documentation

BITMAP ENCODING

Store a separate bitmap for each unique value for an attribute where an offset in the vector corresponds to a tuple.

- → The i^{th} position in the Bitmap corresponds to the i^{th} tuple in the table.
- → Typically segmented into chunks to avoid allocating large blocks of contiguous memory.

Only practical if the value cardinality is low.

Some DBMSs provide bitmap indexes.

BITMAP ENCODING

Original Data

id	sex
1	M
2	M
3	M
4	F
6	M
7	F
8	M
9	M

Compressed Data

id	sex	
	M	F
1	1	0
2	1	0
3	1	0
4	0	1
6	1	0
7	0	1
8	1	0
9	1	0

BITMAP ENCODING

Original Data

id	sex
1	M
2	M
3	M
4	F
6	M
7	F
8	M
9	M

9 × 8-bits = 72 bits

Compressed Data

id	sex	
	M	F
1	1	0
2	1	0
3	1	0
4	0	1
6	1	0
7	0	1
8	1	0
9	1	0

2 × 8-bits = 16 bits

9 × 2-bits = 18 bits

BITMAP ENCODING: EXAMPLE

Assume we have 10 million tuples.
43,000 zip codes in the US.

→ **10000000 × 32-bits = 40 MB**

→ **10000000 × 43000 = 53.75 GB**

Every time the application inserts a new tuple, the DBMS must extend 43,000 different bitmaps.

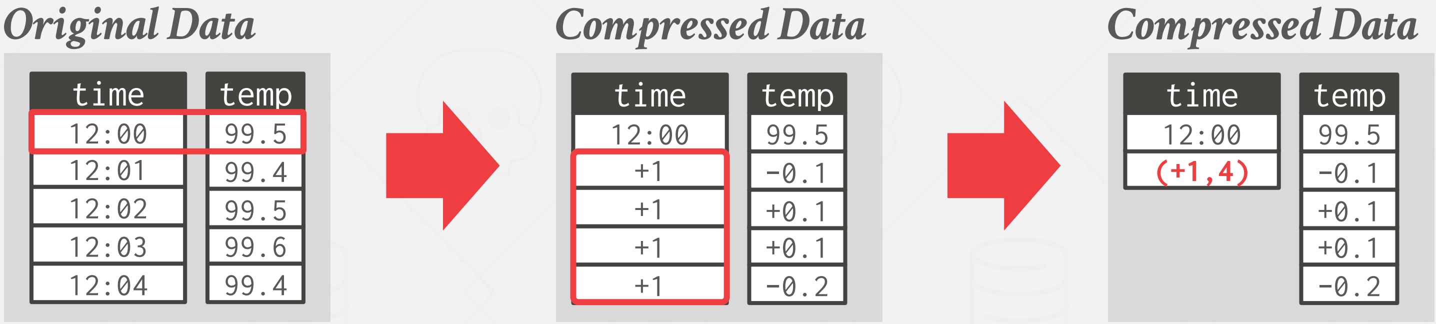
```
CREATE TABLE customer_dim (
  id INT PRIMARY KEY,
  name VARCHAR(32),
  email VARCHAR(64),
  address VARCHAR(64),
  zip_code INT
);
```

DELTA ENCODING

Recording the difference between values that follow each other in the same column.

→ Store base value in-line or in a separate look-up table.

→ Combine with RLE to get even better compression ratios.



Original Data

time	temp
12:00	99.5
12:01	99.4
12:02	99.5
12:03	99.6
12:04	99.4

Compressed Data

time	temp
12:00	99.5
+1	-0.1
+1	+0.1
+1	+0.1
+1	-0.2

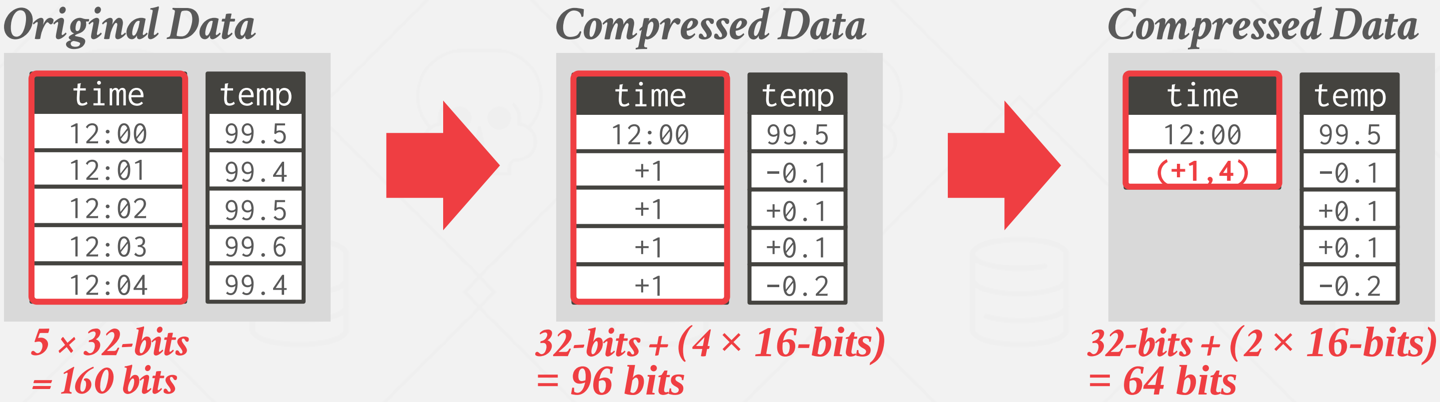
Compressed Data

time	temp
12:00	99.5
(+1,4)	-0.1
	+0.1
	+0.1
	-0.2

DELTA ENCODING

Recording the difference between values that follow each other in the same column.

- → Store base value in-line or in a separate look-up table.
- → Combine with RLE to get even better compression ratios.



Original Data

time	temp
12:00	99.5
12:01	99.4
12:02	99.5
12:03	99.6
12:04	99.4

5 × 32-bits
= 160 bits

Compressed Data

time	temp
12:00	99.5
+1	-0.1
+1	+0.1
+1	+0.1
+1	-0.2

32-bits + (4 × 16-bits)
= 96 bits

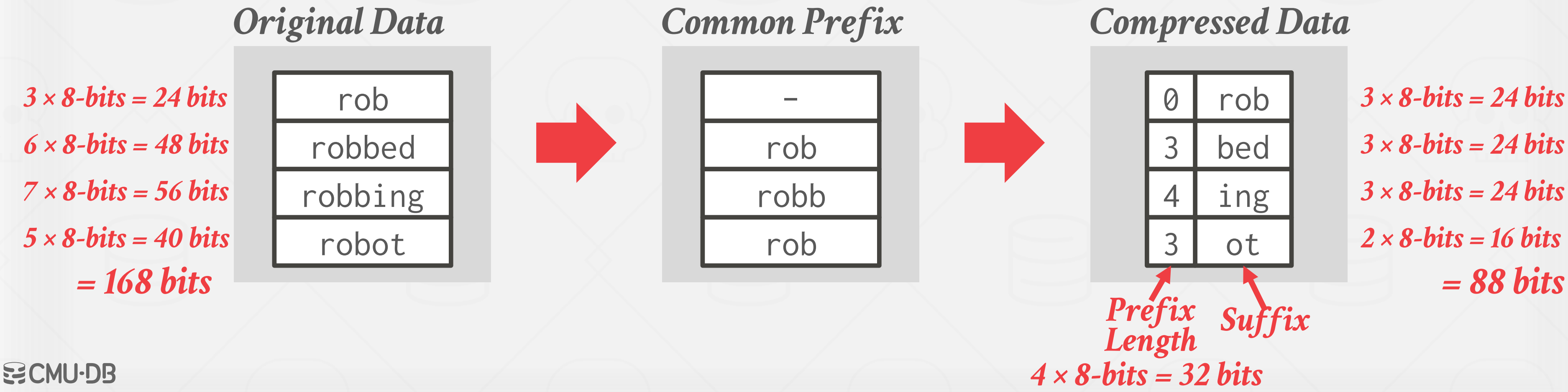
Compressed Data

time	temp
12:00	99.5
(+1,4)	-0.1
	+0.1
	+0.1
	-0.2

32-bits + (2 × 16-bits)
= 64 bits

INCREMENTAL ENCODING

Type of delta encoding that avoids duplicating common prefixes/suffixes between consecutive tuples. This works best with sorted data.



Original Data

3 × 8-bits = 24 bits	rob
6 × 8-bits = 48 bits	robbed
7 × 8-bits = 56 bits	robbing
5 × 8-bits = 40 bits	robot

= 168 bits

Common Prefix

-
rob
robb
rob

Compressed Data

0	rob	*3 × 8-bits = 24 bits*
3	bed	*3 × 8-bits = 24 bits*
4	ing	*3 × 8-bits = 24 bits*
3	ot	*2 × 8-bits = 16 bits*

= 88 bits

Prefix Length

Suffix

4 × 8-bits = 32 bits

DICTIONARY COMPRESSION

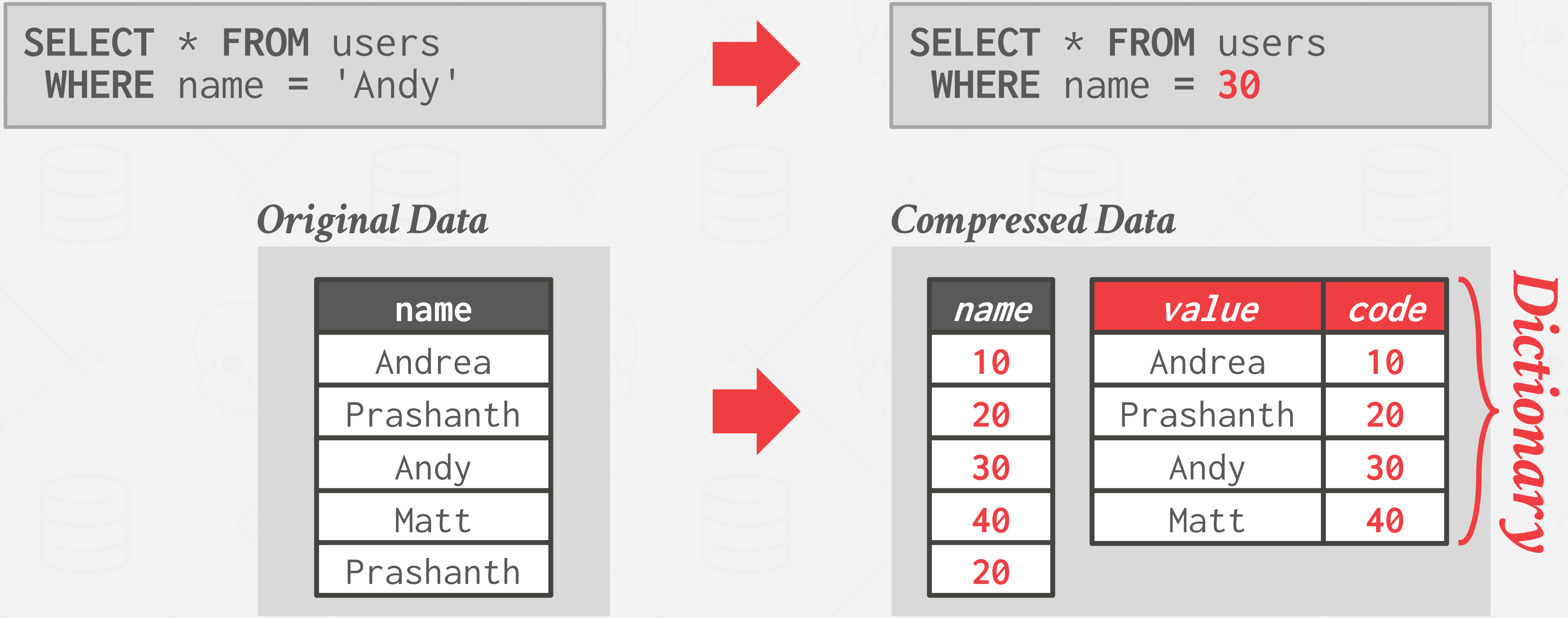
Build a data structure that maps variable-length values to a smaller integer identifier.

Replace those values with their corresponding identifier in the dictionary data structure.

→ Need to support fast encoding and decoding.
→ Need to also support range queries.

Most widely used compression scheme in DBMSs.

DICTIONARY COMPRESSION



```
SELECT * FROM users
 WHERE name = 'Andy'
```

```
SELECT * FROM users
 WHERE name = 30
```

Original Data

name
Andrea
Prashanth
Andy
Matt
Prashanth

Compressed Data

name
10
20
30
40
20

value	*code*
Andrea	10
Prashanth	20
Andy	30
Matt	40

Dictionary

ENCODING / DECODING

A dictionary needs to support two operations:

- → **Encode/Locate:** For a given uncompressed value, convert it into its compressed form.
- → **Decode/Extract:** For a given compressed value, convert it back into its original form.

No magic hash function will do this for us.

ORDER-PRESERVING ENCODING

The encoded values need to support the same collation as the original values.

```
SELECT * FROM users
 WHERE name LIKE 'And%'
```

```
SELECT * FROM users
 WHERE name BETWEEN 10 AND 20
```

Original Data

name
Andrea
Prashanth
Andy
Matt
Prashanth

Compressed Data

name
10
40
20
30
40

value	*code*
Andrea	10
Andy	20
Matt	30
Prashanth	40

Sorted Dictionary

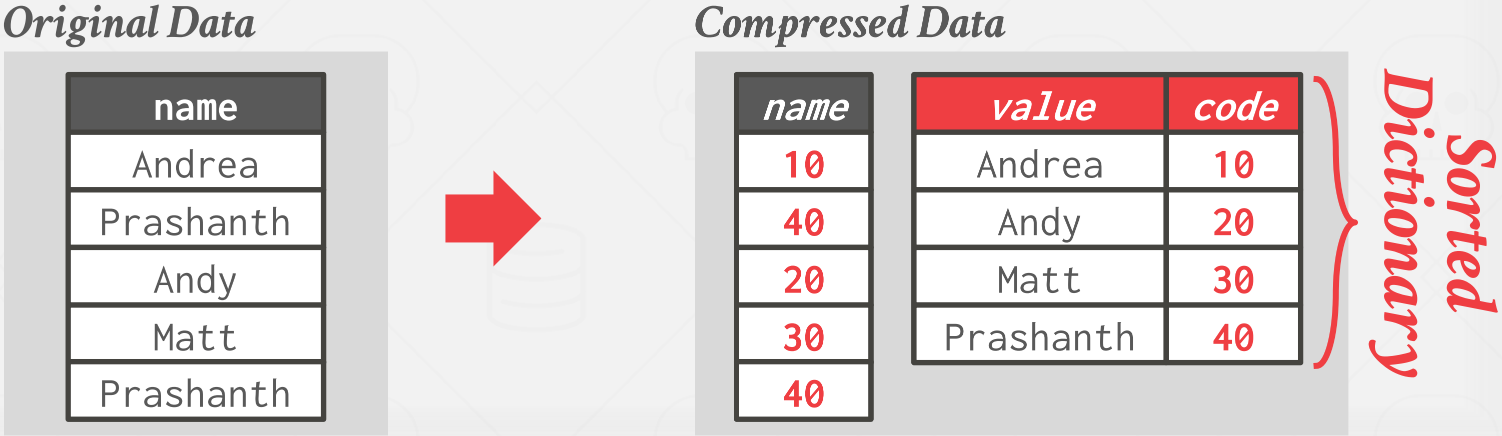
ORDER-PRESERVING ENCODING

```
SELECT name FROM users
 WHERE name LIKE 'And%'
```

Still must perform scan on column

```
SELECT DISTINCT name
  FROM users
 WHERE name LIKE 'And%'
```

Only need to access dictionary



Original Data

name
Andrea
Prashanth
Andy
Matt
Prashanth

Compressed Data

name
10
40
20
30
40

value	*code*
Andrea	10
Andy	20
Matt	30
Prashanth	40

Sorted Dictionary

CONCLUSION

It is important to choose the right storage model for the target workload:

- → OLTP = Row Store
- → OLAP = Column Store

DBMSs can combine different approaches for even better compression.

Dictionary encoding is probably the most useful scheme because it does not require pre-sorting.

DATABASE STORAGE

Problem #1: How the DBMS represents the database in files on disk.

Problem #2: How the DBMS manages its memory and move data back-and-forth from disk.

← Next