Introduction to
In-memory Column-based Databases

Radim Benek, SAP Labs Czech Republic
2022

PUBLIC

THE BEST RUN SAP

SAP Labs Czech Republic

This is our story

in #brnoregion

Who we are

Total Employees
450

Millenials
65%

Women in Tech
18%

Early Talents
three year post-grad
15%

Average Tenure
6.9 years

Embrace differences
23 nationalities

Achievements

Proud to be an **award winning employer** assessed on criteria including work atmosphere, benefits, training and skills development, and community involvement.

- Excellent employee conditions 2021 by **Top Employer**
- Best Places to Work 2021 by **Glassdoor**
- Top 10 Employer in IT 2021 by **TOP Zaměstnavatelé**

What we do

We develop, localize and maintain on our flagship product **SAP S/4HANA** with focus on **Finance** and Sustainability.

Labs, Globalization, Center, Thought, Experience, Solutions, Responsible, Applications, Sustainability, Training, Support, Cloud, Best, Finance, Development, Customers, Efficiently, Friendly, Purpose, Deep, Maintenance, Industries, Core, Innovation, S/4HANA, Leadership, Excellent, Business, Well-being, Front, End-to-end

Our employee groups

- Early Talents
- Mindfulness
- D-shop
- Runners
- Crossfit
- Band

and more...

SAP Labs Czech Republic

Ecosystem

MUNI

BRNO UNIVERSITY OF TECHNOLOGY

Mendel University in Brno

JIC

and more...

Our CSR partners

czechitas

REPLUG me

asociace společenské odpovědnosti

MALÁ TECHNICKÁ UNIVERZITA

Our history

- 1995 — SAP in Brno established
- 2007 — Moving to Spielberk Office Centre
- 2008 — 100 employees
- 2016 — Status of Labs received SAP Labs
- 2020 — 25 Anniversary

Stay in touch

Holandská 2/4
Brno 639 00
Czech Republic

+420 543 542 411

labs.cz@sap.com

SAP Labs CZ

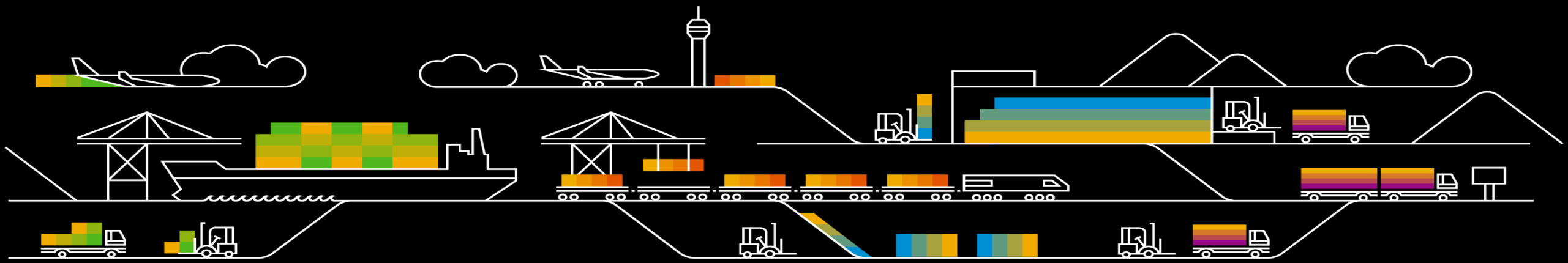
Join us: **www.sap.cz/careers-brno**

last update November 2021

Agenda

- Introduction
- Changes in Hardware
- Data Layout
- Dictionary Encoding
- Compression
- Delete, Insert, Update
- Tuple Reconstruction
- Scan Performance
- Demo

Introduction

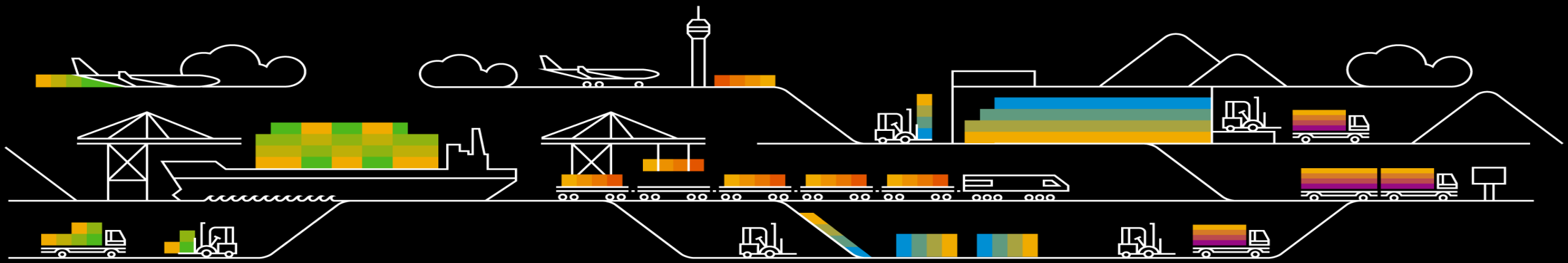


Introduction

Achievements of column store in-memory database:

- 150 sensors
- 2GB of data in one lap
- **3TB** in a single race
- „SAP HANA enables McLaren existing systems to process these data **14 000** times faster then before.“
- „Analysis that previously took almost a **week** to process, can be completed in a span of a **pit stop**.“

Changes in Hardware



Changes in Hardware

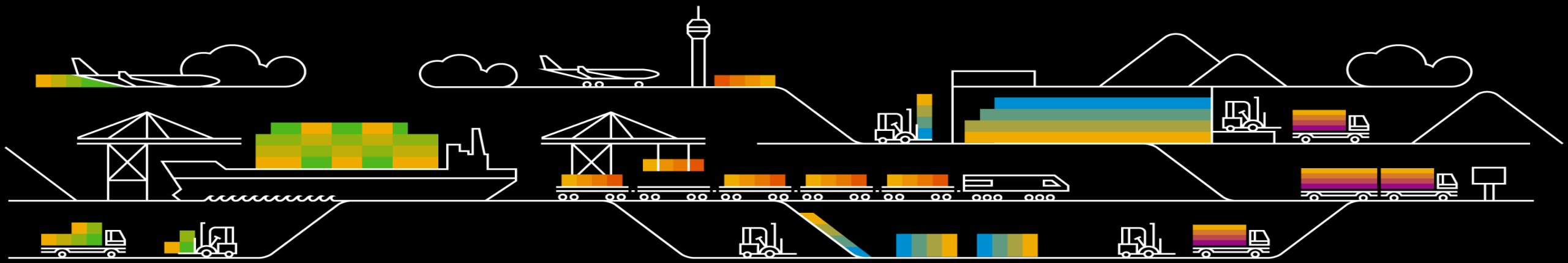
Evolution

- Multi-core CPU introduction (32 cores/CPU)
- Multi-CPU boards massively used (8 CPUs/board)
- CPU cache grows
- RAM capacity grows
- RAM speed grows
- New interfaces (QPI, HT)

→ Enormous bandwidth and performance potential

HDDs still dominated overall performance and design mindset

Data Layout



Data Layout

Database Data Layouts

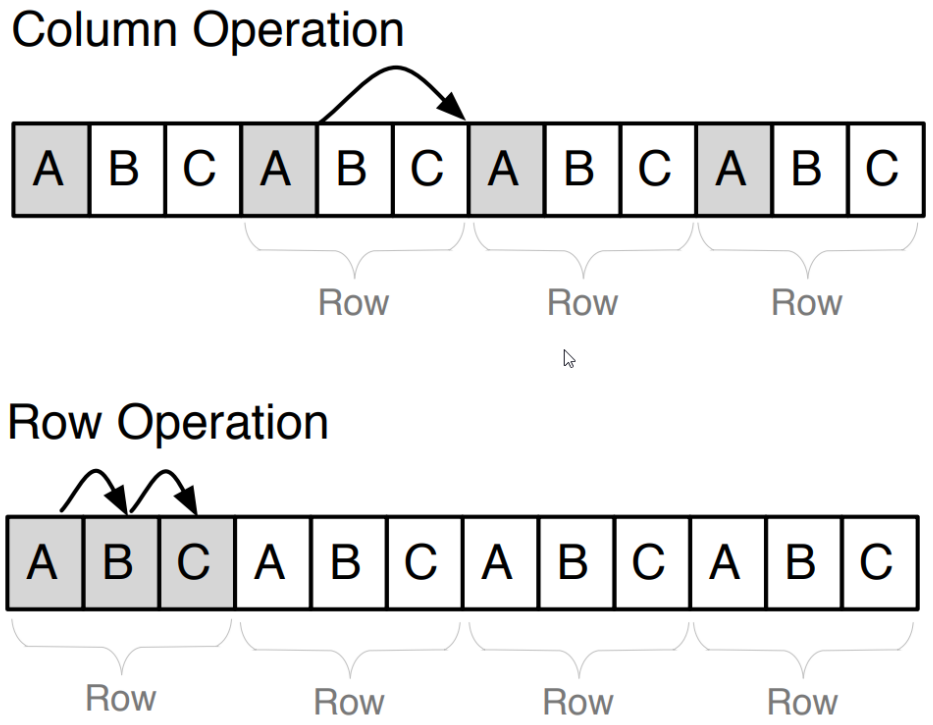
- What are the most common layouts of relational data in main memory?
  – For each layout we present the pros and cons of their approach

	Col_1	Col_2	Col_3
Row_1	A	B	C
Row_2	A	B	C
Row_3	A	B	C
Row_3	A	B	C

Data Layout

Row Data Layouts

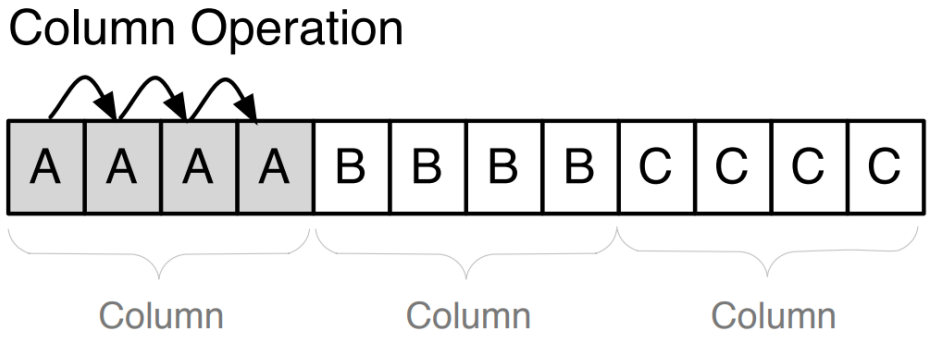
- Data is stored tuple-wise
- Leverage co-location of attributes for a single tuple
- Low cost for reconstruction, but higher cost for sequential scan of a single attribute

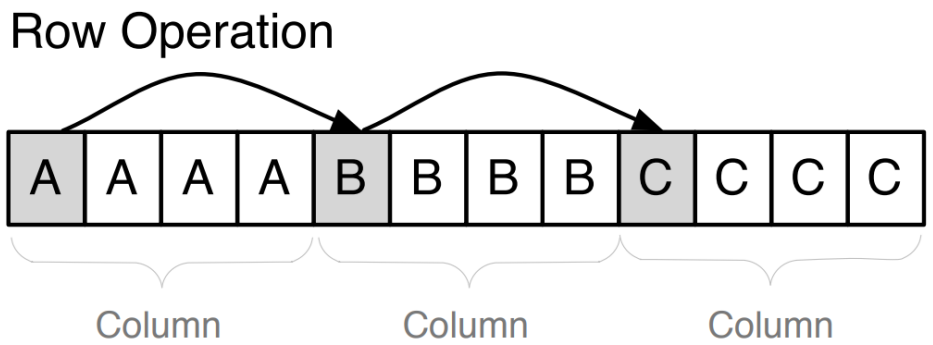


Data Layout

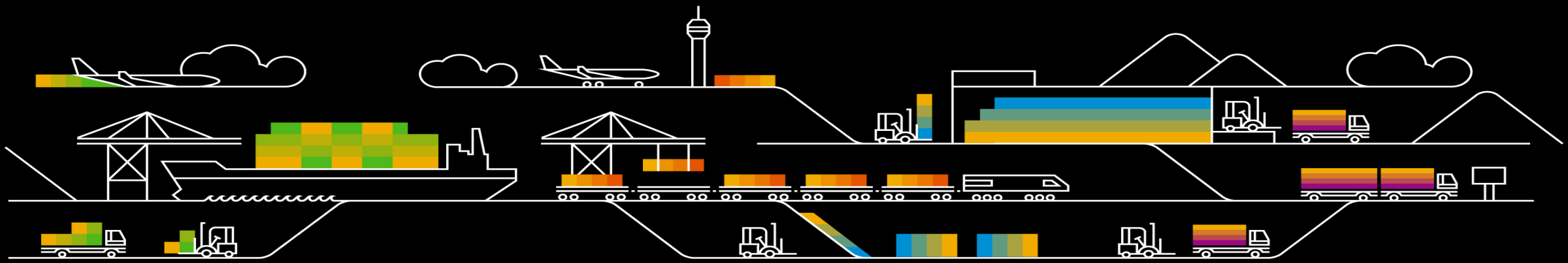
Columnar Data Layouts

- Data is stored attribute-wise
- Leverage sequential scan-speed in main memory
- Tuple reconstruction is expensive





Dictionary Encoding



Dictionary Encoding
Example

- 8 billion humans

- Attributes:
  - first name
  - last name
  - gender
  - country
  - city
  - birthday

  → 200 byte per tuple

- Each attribute is dictionary encoded

Dictionary Encoding

Motivation

- Main memory access is the new bottleneck
- Compression reduces number of I/O operations to main memory
- Operation directly on compressed data
- Offsetting with bit-encoded fixed-length data types
- Based on limited value domain

Dictionary Encoding

Sample Data

Table: world_population

recID	fname	lname	gender	city	country	birthday
...	...	...	...	...	...	...
39	John	Smith	m	Chicago	USA	12.03.1964
40	Mary	Brown	f	London	UK	12.05.1964
41	Jane	Doe	f	Palo Alto	USA	23.04.1976
42	John	Doe	m	Palo Alto	USA	17.06.1952
43	Peter	Schmidt	m	Potsdam	GER	11.11.1975
...	...	...	...		...	...

Dictionary Encoding

Dictionary Encoding a Column

- A column is split into a dictionary and an attribute vector
- Dictionary stores all distinct values with implicit valueID
- Attribute vector stores valueIDs for all entries in the column
- Position is stored implicitly
- Enables offsetting with bit-encoded fixed-length data types

recID	fname
...	...
39	John
40	Mary
41	Jane
42	John
43	Peter
...	...

Dictionary for "fname"

valueID	Value
...	...
23	John
24	Mary
25	Jane
26	Peter
...	...

Attribute Vector for "fname"

position	valueID
...	...
39	23
40	24
41	25
42	23
43	26
...	...

Dictionary Encoding

Querying Data using Dictionaries

Search for Attribute Value
(i.e. retrieve all persons with fname “Mary”)

1. Search valueIDs for requested value (“Mary”)
2. Scan Attribute Vector for valueID (“24”)
3. Replace valueIDs in result with corresponding dictionary value

Dictionary Encoding

Sorted Dictionary

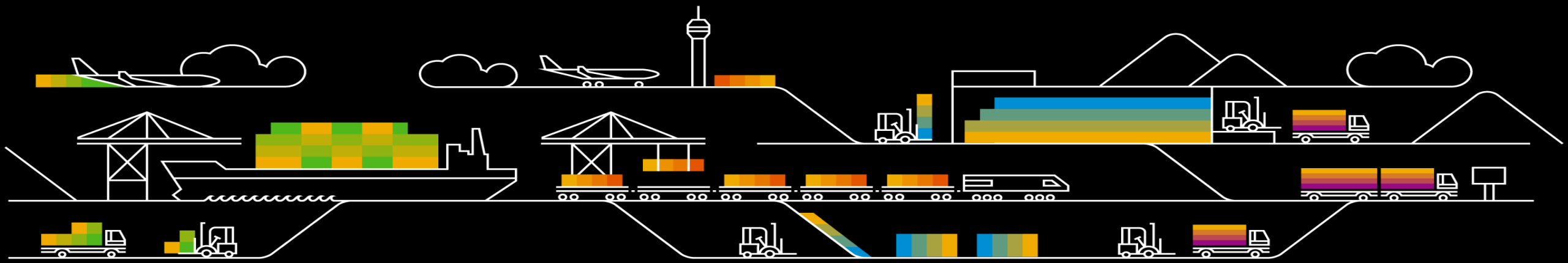
- Dictionary entries are sorted either by their numeric value or lexicographically
  – Dictionary lookup complexity: O(log(n)) instead of O(n)
- Dictionary entries can be compressed to reduce the amount of required storage

Dictionary Encoding

Data Size Examples

Column	Cardi-nality	Bits Needed	Item Size	Plain Size	Size with Dictionary (Dictionary + Column)	Compression Factor
First names	5 millions	23 bit	50 Byte	400GB	250MB + 23GB	≈ 17
Last names	8 millions	23 bit	50 Byte	400GB	400MB + 23GB	≈ 17
Gender	2	1 bit	1 Byte	8GB	2b + 1GB	≈ 8
City	1 million	20 bit	50 Byte	400GB	50MB + 20GB	≈ 20
Country	200	8 bit	47 Byte	376GB	9.4kB + 8GB	≈47
Birthday	40000	16 bit	2 Byte	16GB	80kB + 16GB	≈ 1
Totals			**200 Byte**	**≈ 1.6TB**	**≈ 92GB**	**≈ 17**

Compression



Compression

Compression Techniques

- Heavy weight vs. light weight techniques
- Focus on light weight techniques for databases

- For attribute vector
  - Prefix encoding
  - Run length encoding
  - Cluster encoding
  - Sparse encoding
  - Indirect encoding

- For dictionary
  - Delta compression for strings
  - Other data types are stored as sorted arrays

Compression

Example Table

recID	fname	lname	gender	country	city	birthday	2nd_nationality
0	Martin	Albrecht	m	GER	Berlin	08-05-1955	n/a
1	Michael	Berg	m	GER	Berlin	03-05-1970	n/a
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968	n/a
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992	US
4	Ulrike	Schulze	f	GER	Potsdam	09-03-1977	n/a
5	Martin	Schulz	m	GER	Mainz	06-04-1980	GER
6	Sushi	Pao	f	CN	Peking	09-12-1954	n/a
7	Chen	Su Wong	m	CN	Shanghai	27-06-1999	n/a
...	...	...	...	...	...	...	...

- 200 countries = 8 bit
- 1 million cities = 20 bit
- 100 different 2nd nationalities = 7 bit
- 5 million first names = 23 bit

Compression
Prefix Encoding

- Used if the column starts with a long sequence of the same value
- One predominant value in a column and the remaining values are mostly unique or have low redundancy

Example: country column, table sorted by population of country

Dictionary encoded attribute vector **7.45 GB**

37	…	37	74	…	74	195	…	195	…	197

8 bit

37	74	…	74	195	…	195	…	197

8 bit 8 bit …

1.4B
# of occurrences of first valueID (64 bit)

Using China saves 1.4 billion × 8 bit:
(8 billion – 1.4 billion) × 8 bit + 64 bit + 8 bit
≈ 6.15 GB

Direct access!

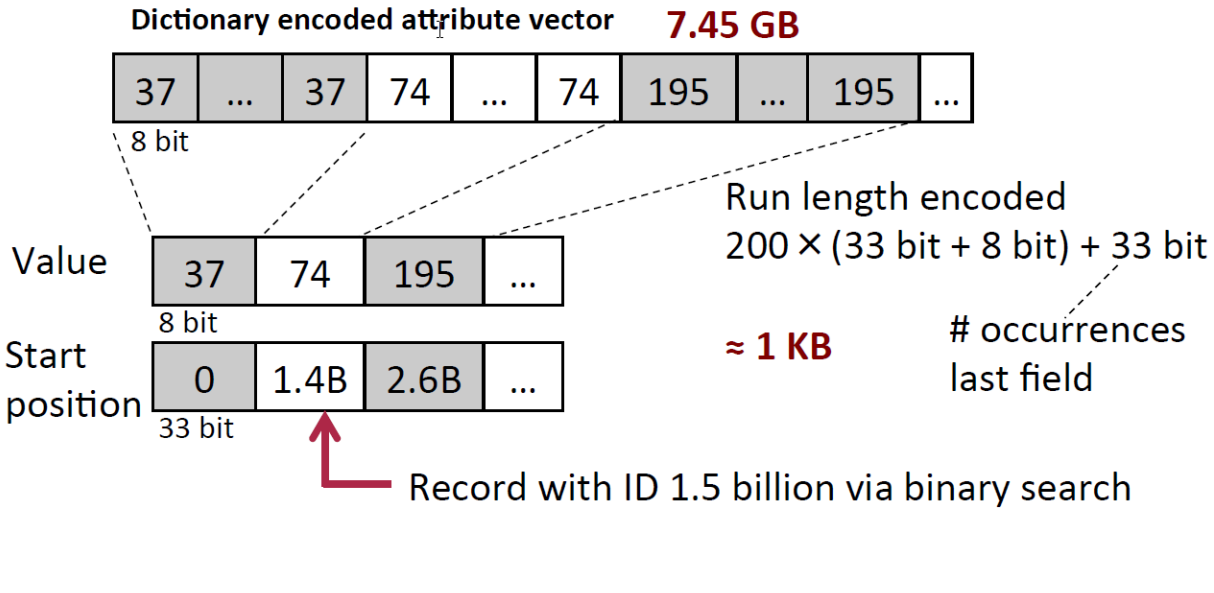
Dictionary

valueID	value
…	…
37	CN
…	…
68	GER
…	…
74	IN
…	…
195	US
…	…
197	VA

Compression
Run Length Encoding

- Replace sequence of the same value with a single instance of the value and
  a. Its number of occurrences
  b. Its start position (shown below)
- Variant b) speeds up access compared to a)

Direct access!

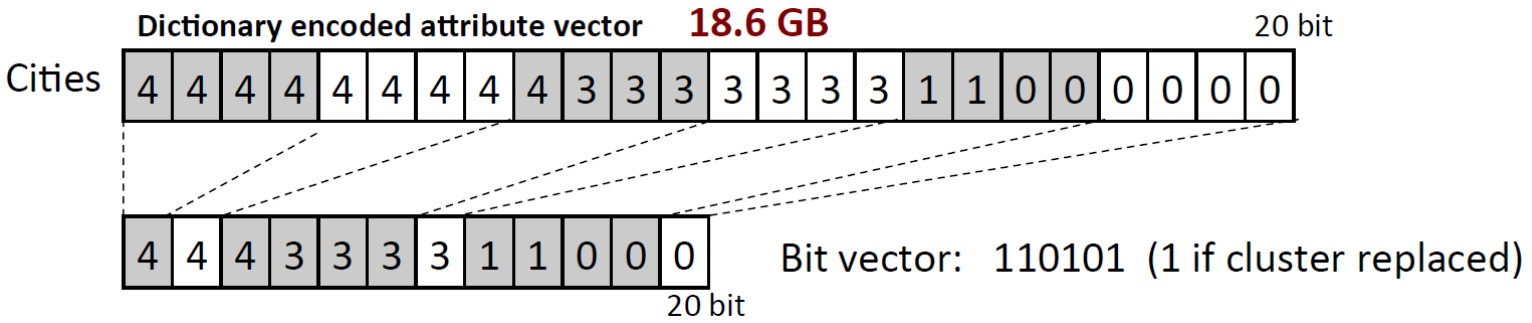


Dictionary

valueID	value
...	...
37	CN
...	...
68	GER
...	...
74	IN
...	...
195	US
...	...

Compression
Cluster Encoding

- Attribute vector is partitioned into *N* blocks of fixed size (typically 1024)
- If a cluster contains only a single value, it is replaced by a single occurrence of this value
- A bit vector of length *N* indicates which clusters were replaced by a single value



- Example: city column, table sorted by country, city
  - Cluster size: 1024 elements →7.8 mio blocks
  - Worst case assumption: 1 uncompressible block per city
  - Uncompressible blocks: 1 mio × 1024 × 20 bit — 2441 MB
  - Compressible blocks: (7.8 - 1) mio × 20 bit — + 16 MB
  - Bit vector: 7.8 million × 1 bit — + 1 MB — **≈2.4 GB**

No direct access! Compute position via bit vector.

Compression
Sparse Encoding

- Remove the value *v* that appears most often
- A bit vector indicates at which positions *v* was removed from the original sequence

Example: 2nd nationality column, regardless of sorting order of table

Dictionary

valueID	value
0	n/a
...	...
4	CO
...	...
9	GER
...	...
95	US
...	...

Dictionary encoded attribute vector **6.5 GB**

...	0	0	0	9	0	0	0	0	9	0	0	0	0	4	0	95	0	0	0	...

7 bit

v	Sparse encoded					Bit vector
0	...	9	9	4	95	...1110111101111010111...
7 bit	7 bit					

Assumption: 99 % of people do not have a 2nd nationality

Bit vector 8 billion × 1 bit 954 MB

Sparse encoded attribute vector

80 mio × 7 bit + 67 MB

value + 7 bit ≈ **1 GB**

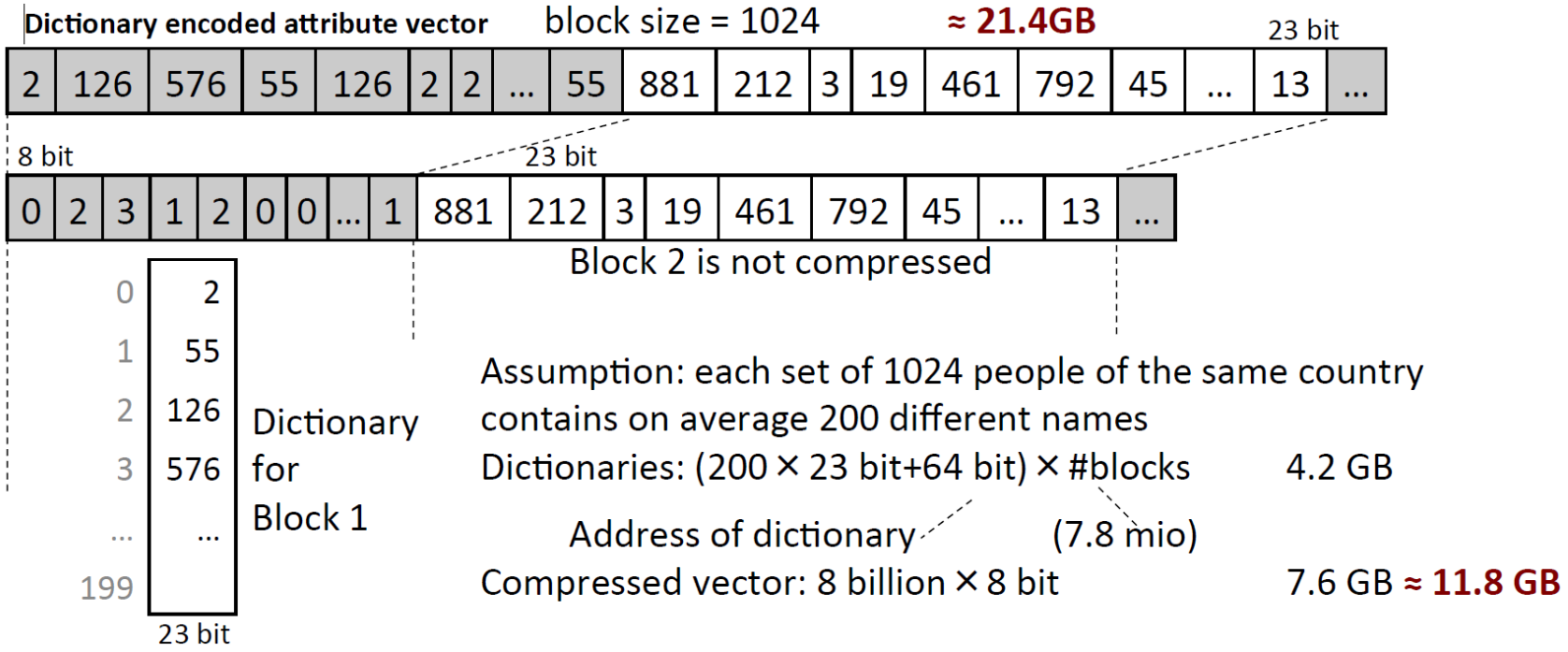
No direct access! Compute position via bit vector.

Compression
Indirect Encoding

- Sequence is partitioned into *N* blocks of size *S* (typically 1024)
- If a block contains only a few distinct values an additional dictionary is used to encode the values in that block
- Additionally: links to the new dictionaries + blocks that have a dictionary

Example: fname column, table sorted by country

Direct access!



Compression

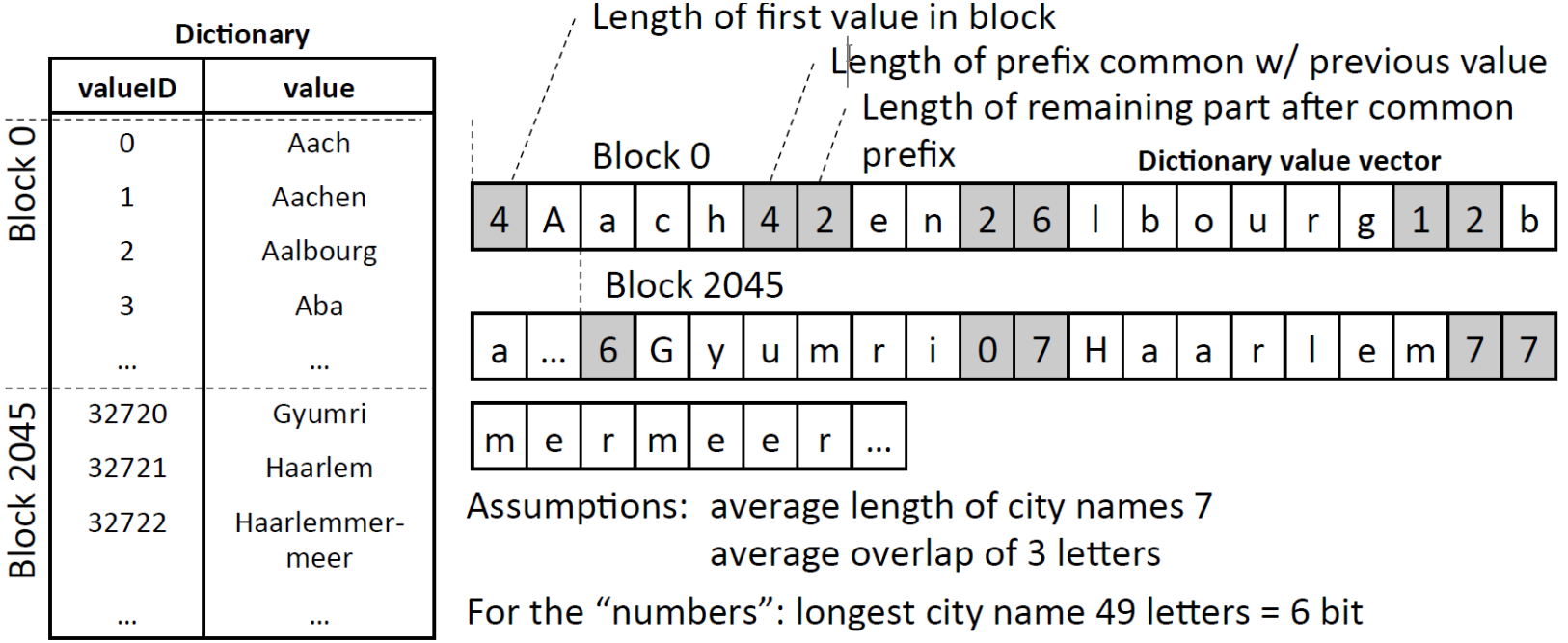
Delta Encoding for Dictionary

- For sorted string values
- Block--wise compression (typically 16 strings per block)

Dictionary:

1 million cities à 49 byte

≈ 46.7 MB



Assumptions: average length of city names 7
average overlap of 3 letters

For the “numbers”: longest city name 49 letters = 6 bit

Size of block × #blocks

(encoding numbers + 1st city + 15 other cities) × #blocks

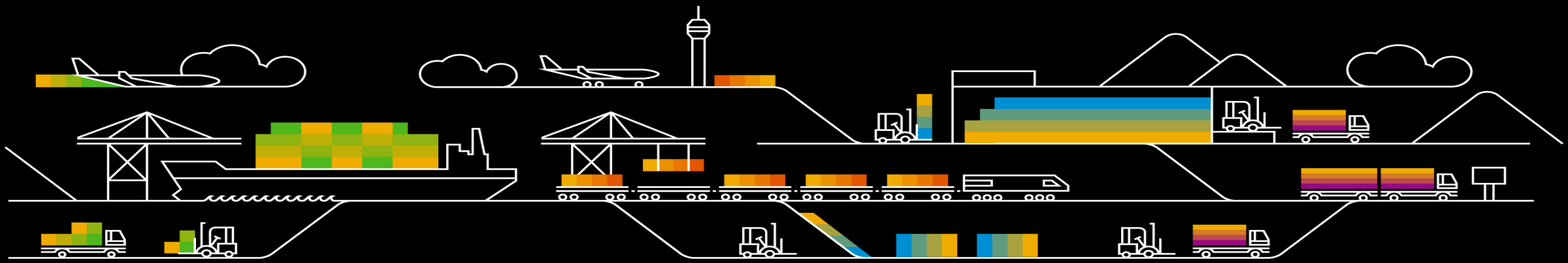
((1+15 × 2) × 6 bit + 7 × 1 byte + 15 × (7-3) × 1 byte) × 62500

≈ 5.4 MB

Compression
Keep in Mind

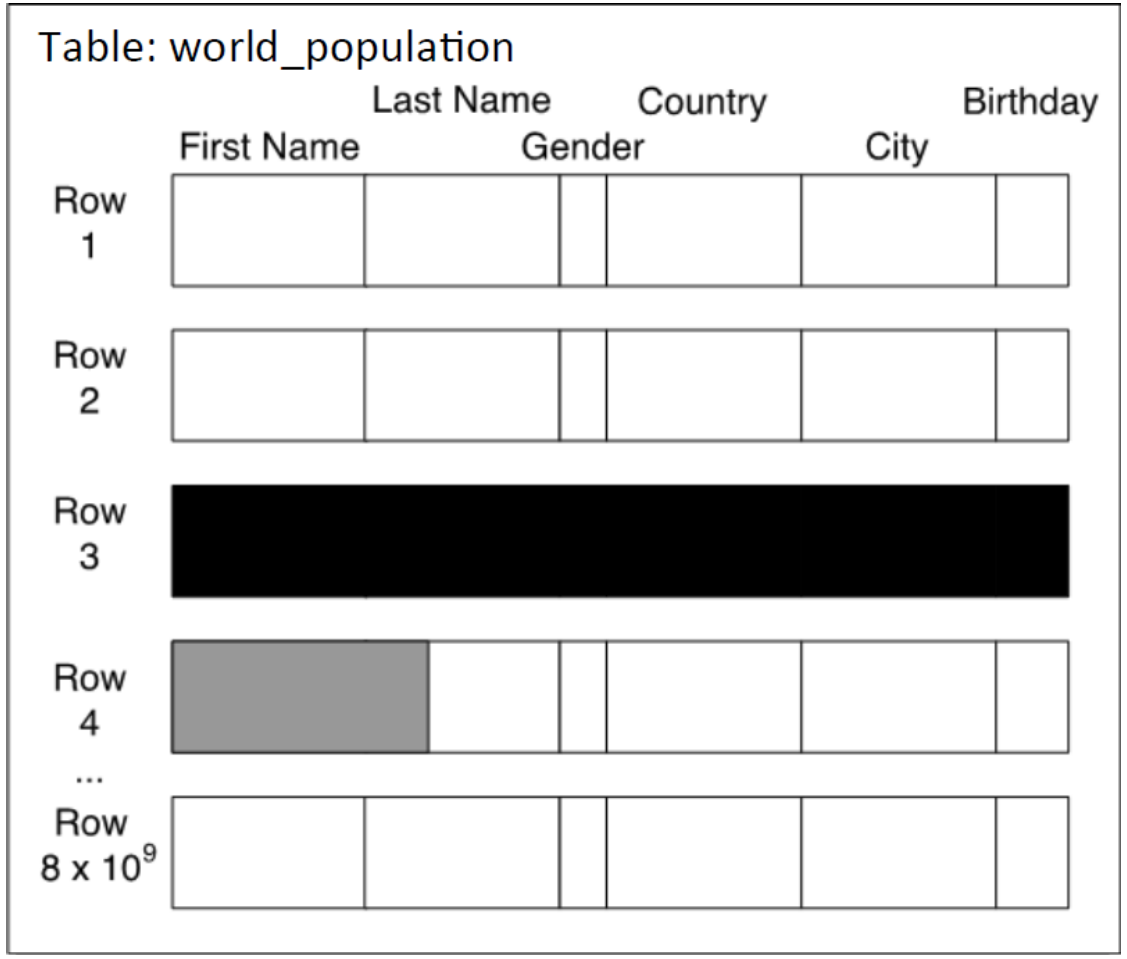
- Most compression techniques require sorted sets, but a table can only be sorted by one column or cascading
- No direct access to rows in some cases, but offset has to be computed

Tuple Reconstruction



Tuple Reconstruction

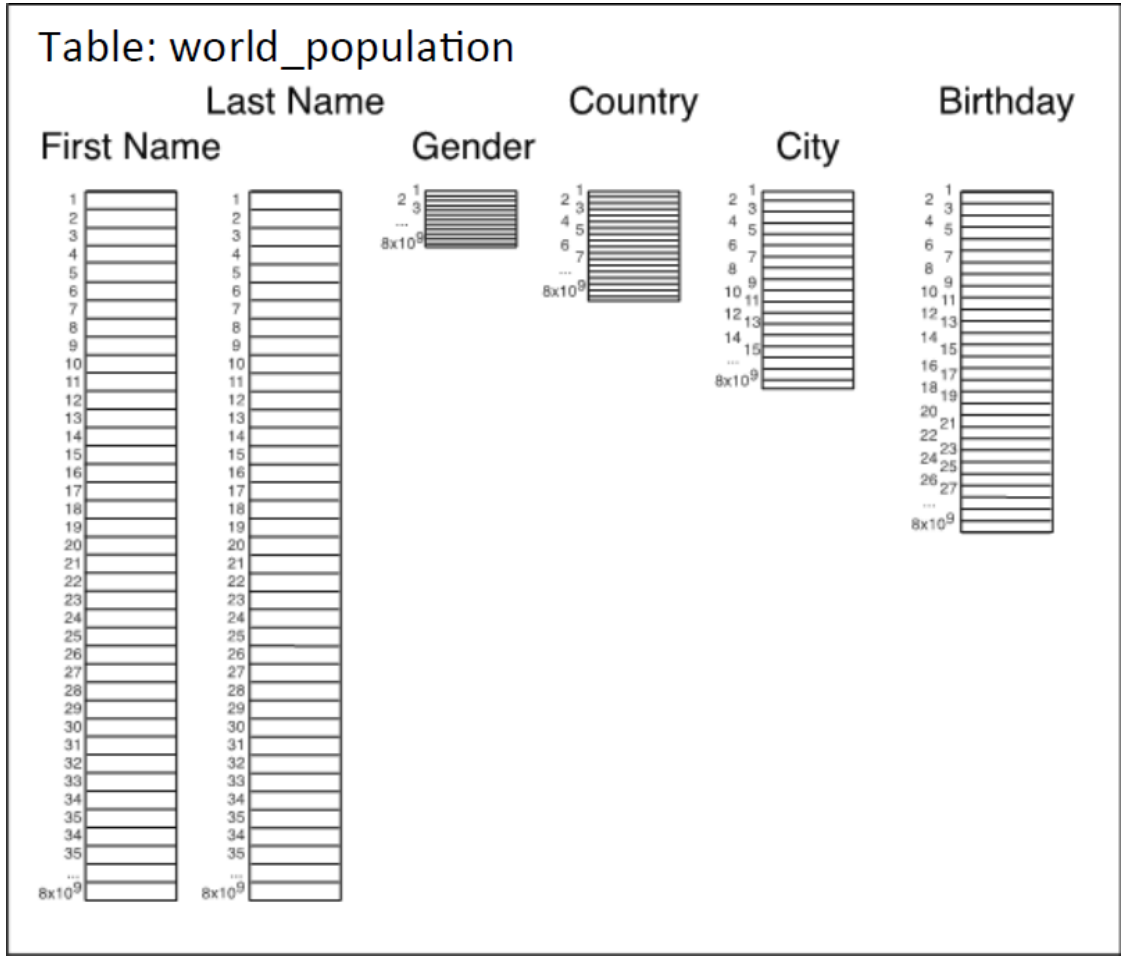
Accessing a record in a row store



- All attributes are stored consecutively
- 200 byte → 4 cache accesses à 64 byte → 256 byte
- Read with 4MB/ms/core
- → ≈ **0.064 µs** with 1 core

Tuple Reconstruction

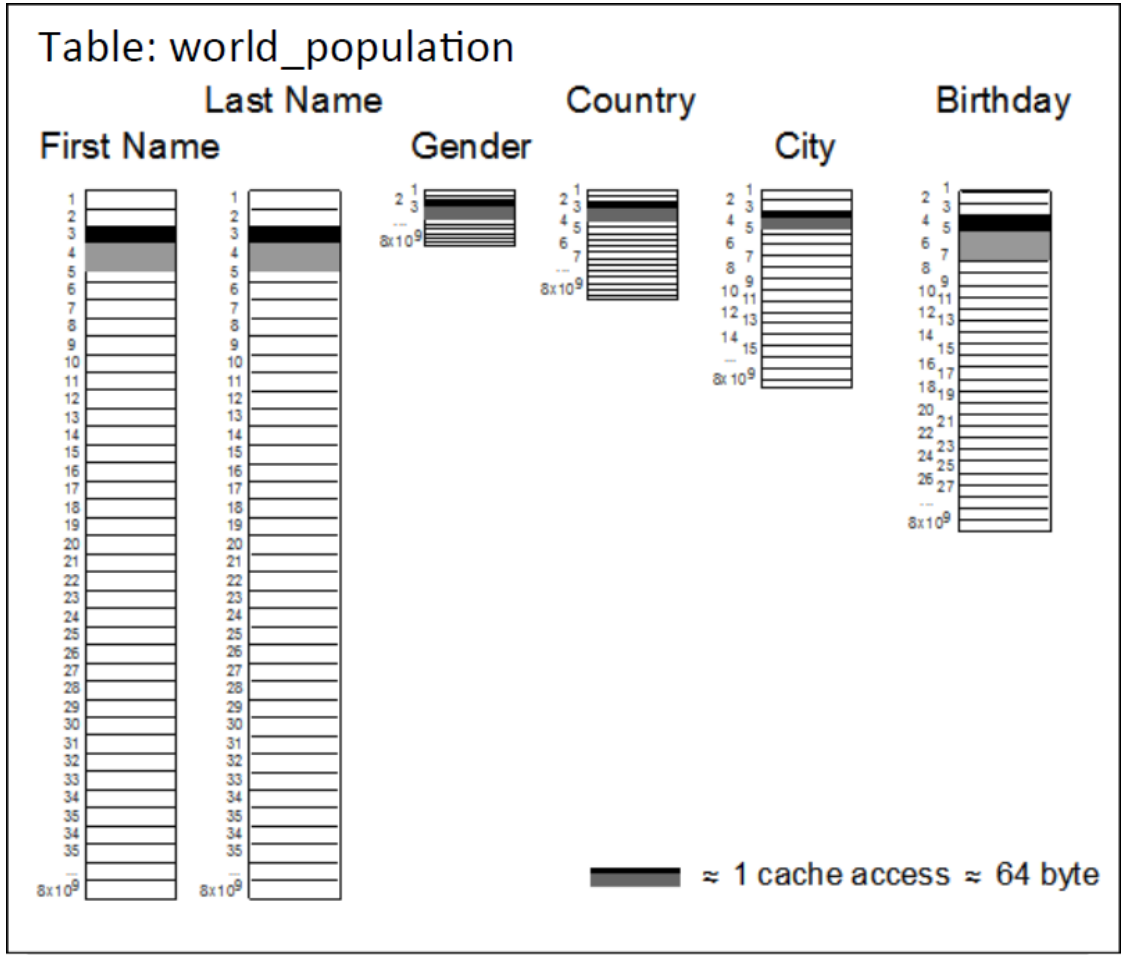
Virtual record IDs



- All attributes are stored in separate columns
- Implicit record Ids are used to reconstruct rows

Tuple Reconstruction

Virtual record IDs

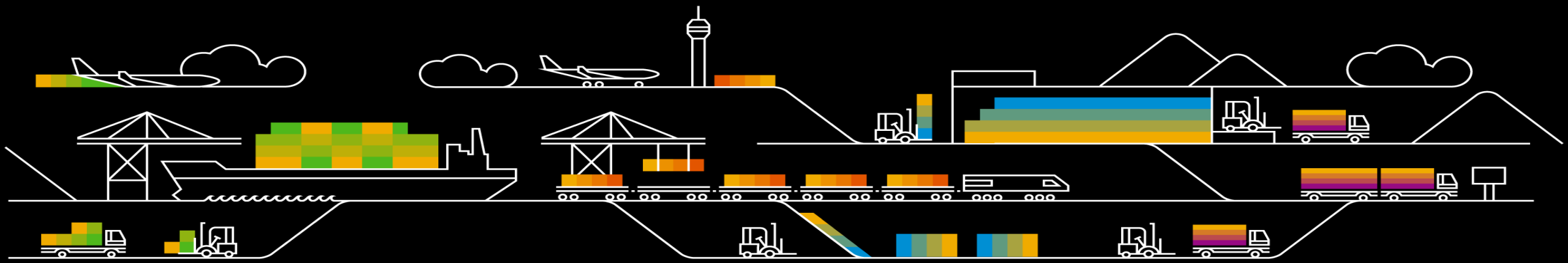


- 1 cache access for each attribute
- 6 cache accesses à 64 byte
  → 384 byte
- Read with 4MB/ms/core
- → ≈ **0.096 µs** with 1 core

Data loaded and used

Data loaded but not used

Scan Performance



Scan Performance

- 8 billion humans

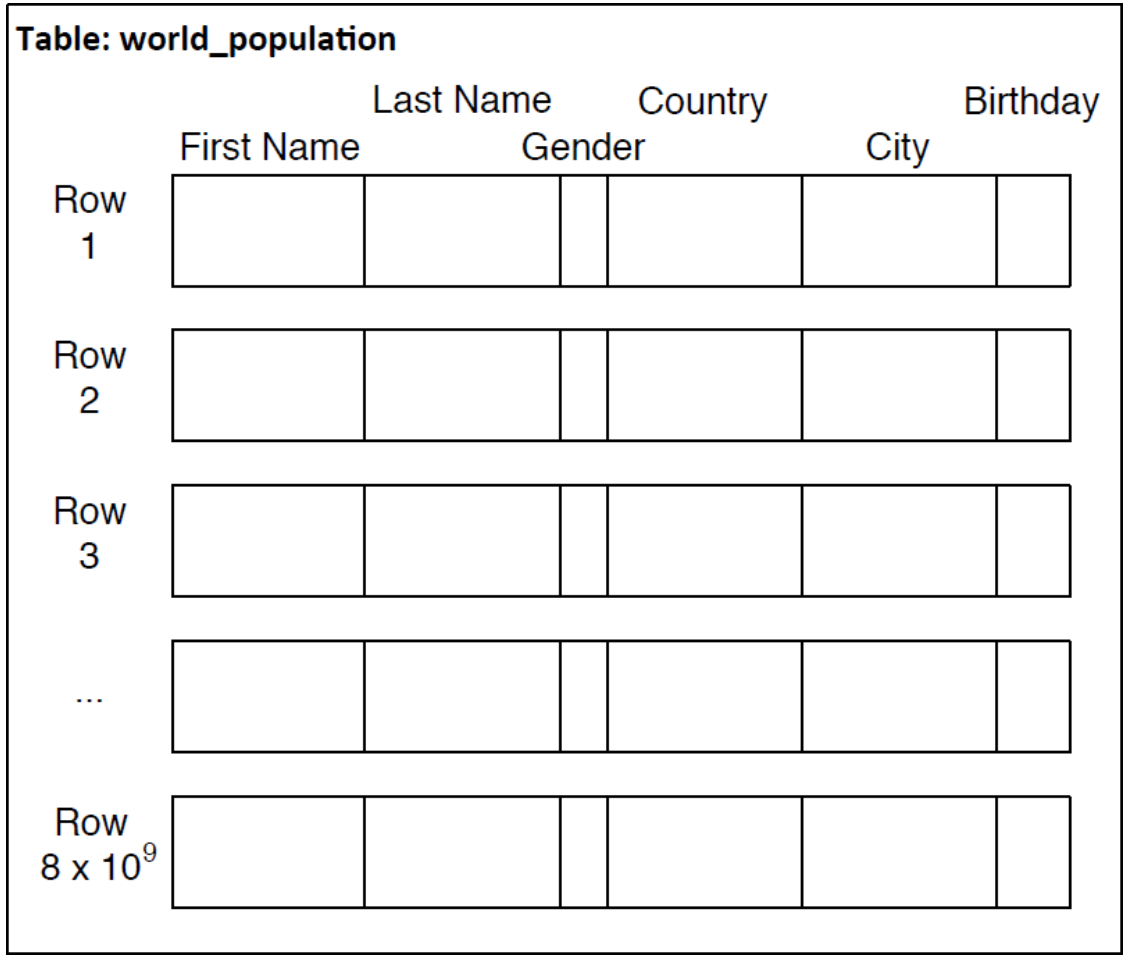
- Attributes:
  - first name
  - last name
  - gender
  - country
  - city
  - birthday

  → 200 byte per tuple

- Question: How many women, how many men?
- Assumed scan speed: 4MB/ms/core

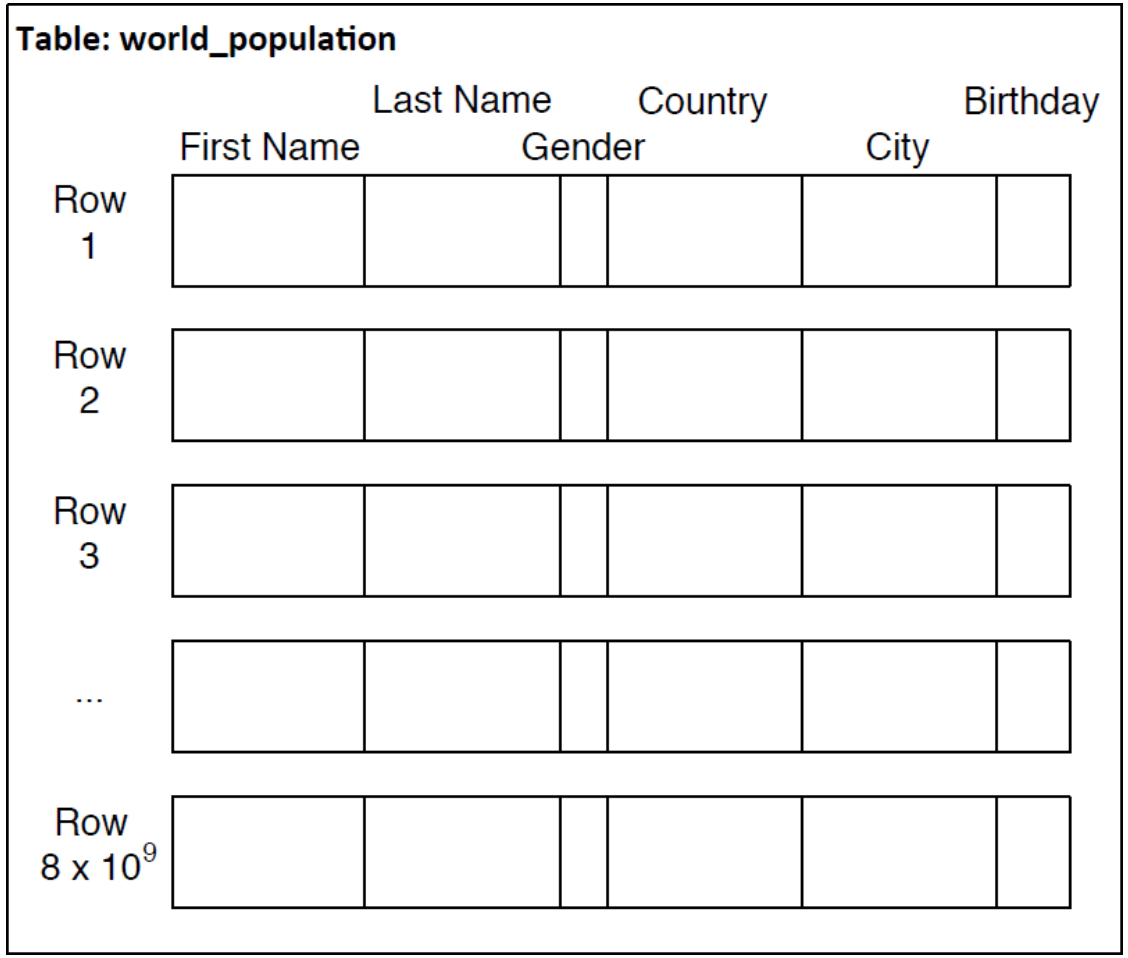
Scan Performance

Row Store – Layout



Scan Performance

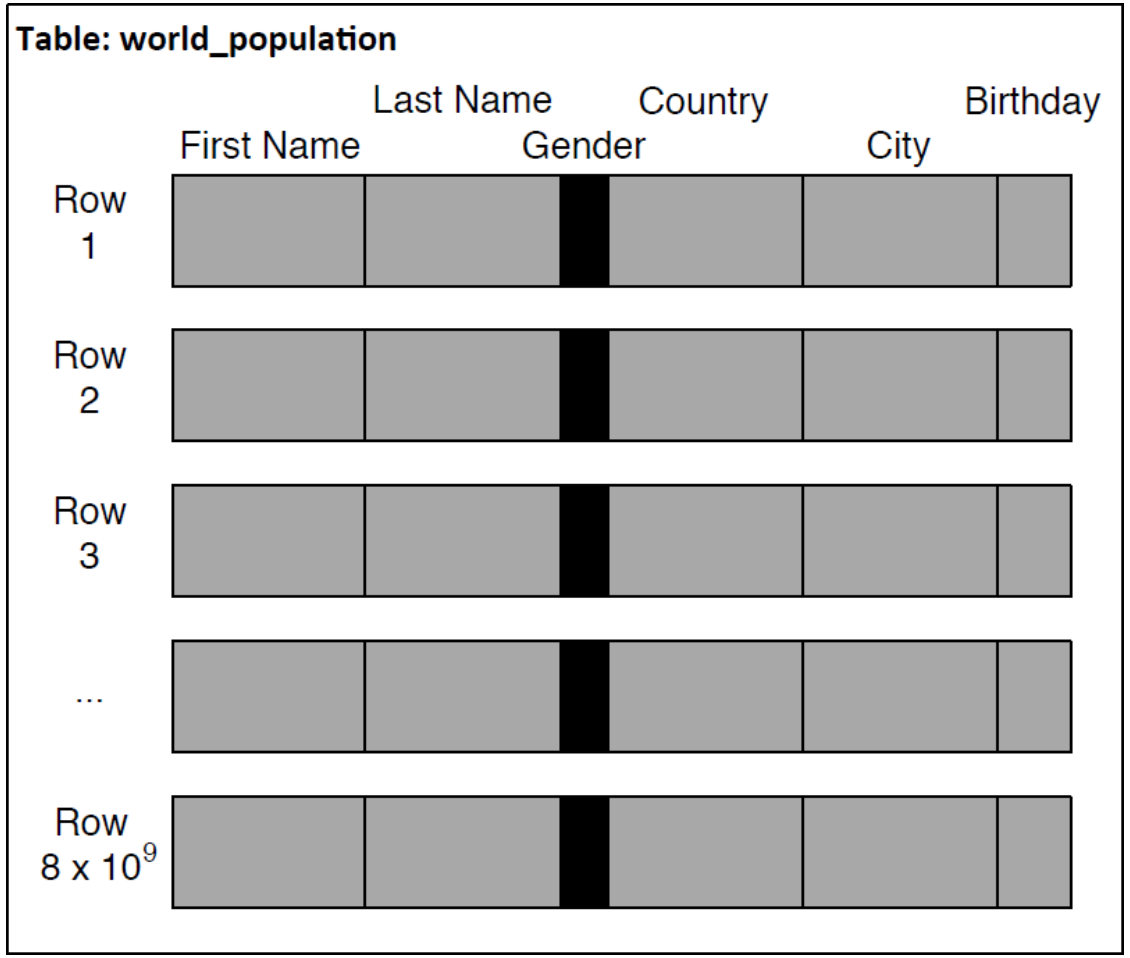
Row Store – Layout



- Table size:
  8 billion tuples × 200 bytes per tuple ≈ **1.6 TB**

Scan Performance

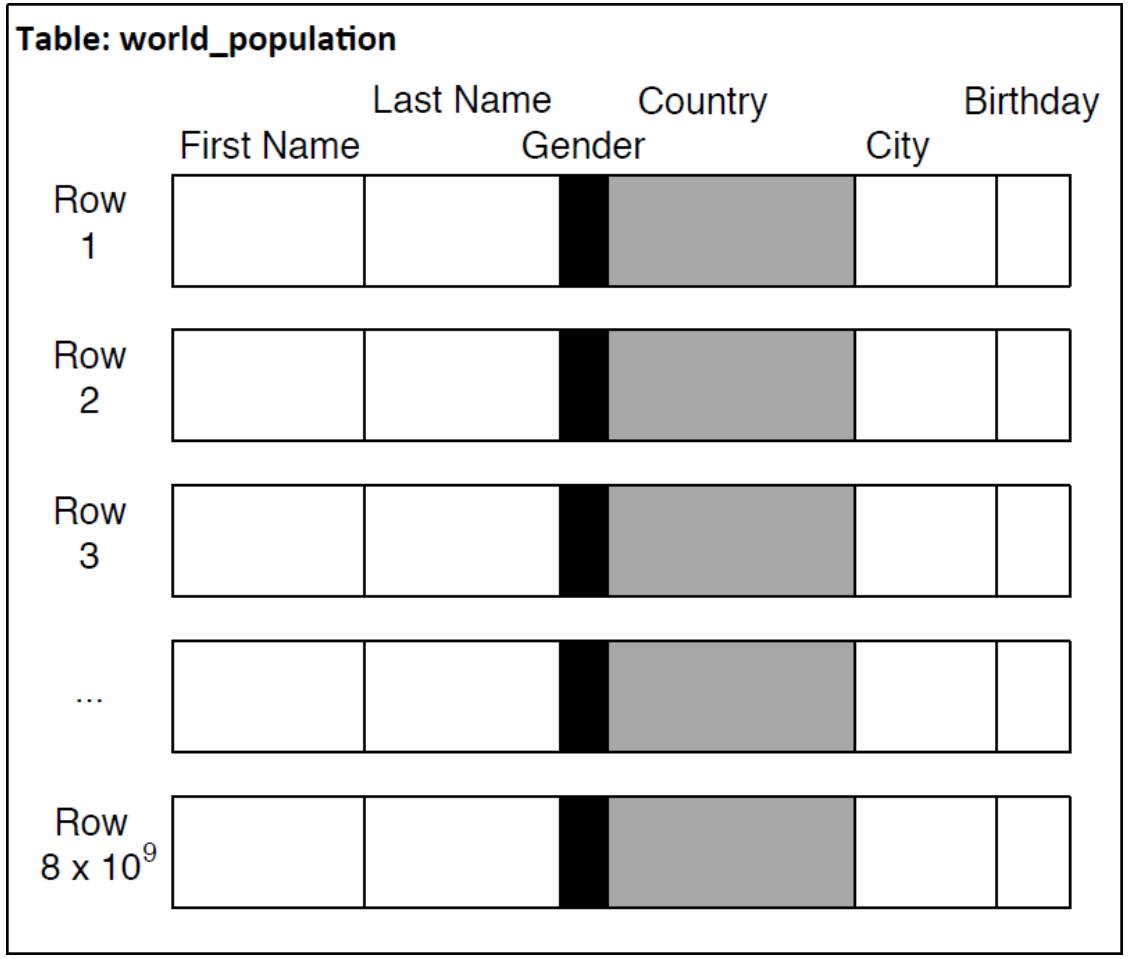
Row Store – Full Table Scan



- Table size:
  8 billion tuples × 200 bytes per tuple **≈ 1.6 TB**
- Scan through all rows with 4MB/ms/core
  → **400 s**
  with 1 core

Scan Performance

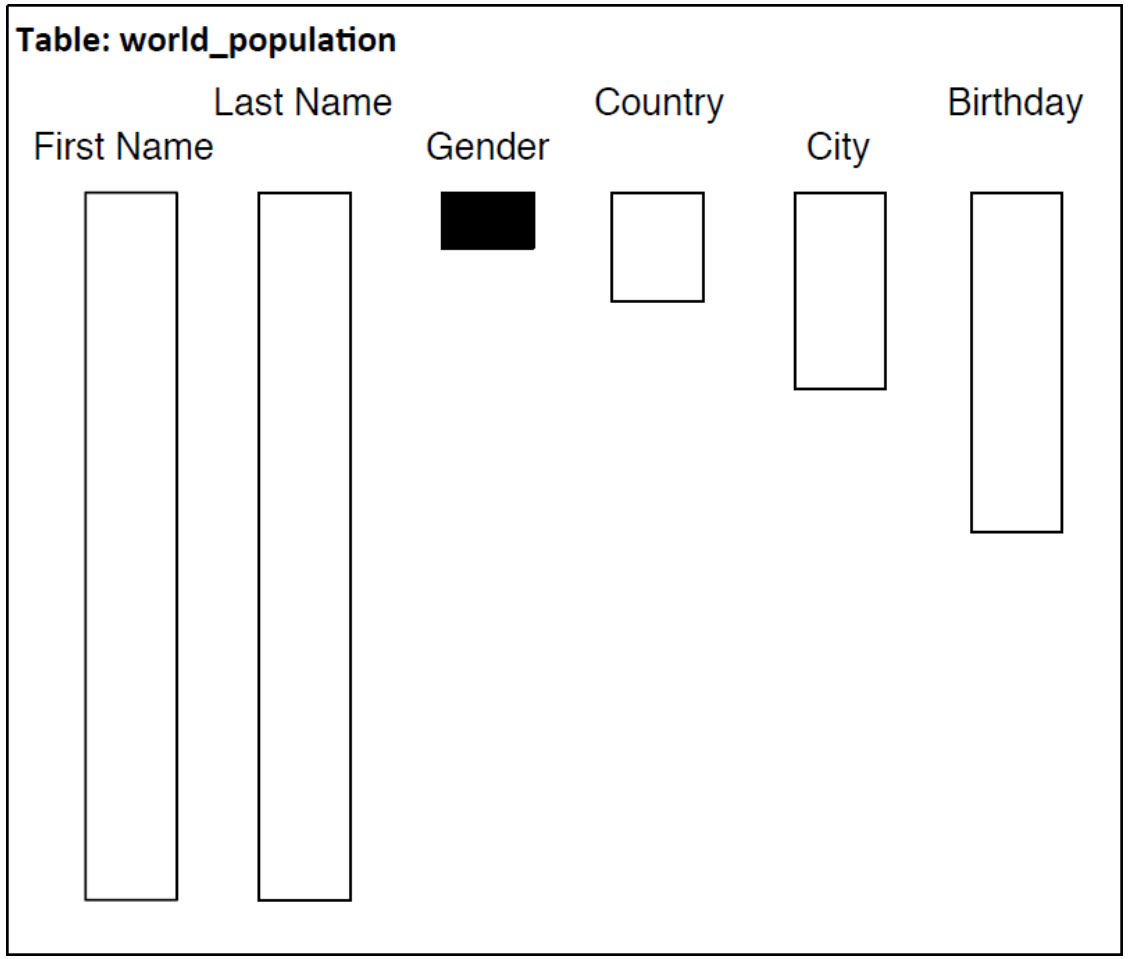
Row Store – Stride Access „Gender“



- 8 billion cache accesses à 64 byte
  ≈ 512 GB
- Read with 4MB/ms/core
  → 128 s
  with 1 core

Scan Performance

Column Store – Full Column Scan „Gender“



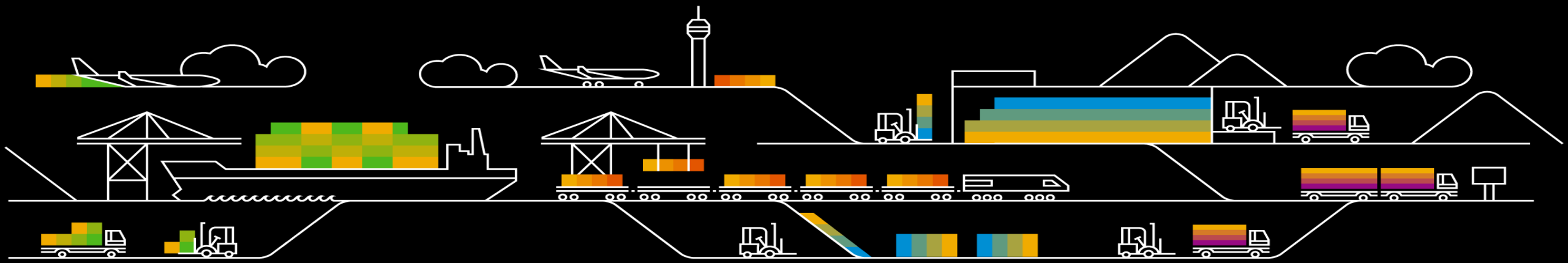
- Size of attribute vector “Gender”:
  8 billion tuples ×
  1 bit per tuple
  ≈ 1 GB
- Scan through column with
  4MB/ms/core
  → **0.25 s** with 1 core

Scan Performance

How many women, how many men?

	Row Store		Column Store
	Full table scan	Stride access	
Time in seconds	400	128	0.25

Delete, Insert, Update



DELETE

Database Operations
DELETE

- Physical DELETE
  – Removed tuple is removed from database and you cannot access it anymore

- Logical DELETE
  – Validity of this tuple is set to non-valid and this tuple can be accessed in historic queries or reporting

- Operation DELETE is very expensive to perform

- SQL-Syntax:

```
DELETE FROM table_name
WHERE attribute_name = some_value
```

Database Operations
DELETE - example

- Remove Jane Doe from the database table

Dictionary “fname”

valueID	value
...	...
22	Andrew
23	Jane
24	John
25	Mary
26	Peter
...	...

Attribute Vector “fname”

recID	valueID
...	...
38	22
39	24
40	25
41	23
42	24
43	26
...	...

Dictionary “lname”

valueID	value
...	...
17	Brown
18	Doe
19	Miller
20	Schmidt
21	Smith
...	...

Attribute Vector “lname”

recID	valueID
...	...
38	19
39	21
40	17
41	18
42	18
43	20
...	...

Database Operations
DELETE - example

- Remove Jane Doe from the database table

Dictionary "fname"

valueID	value
...	...
22	Andrew
23	Jane
24	John
25	Mary
26	Peter
...	...

Attribute Vector "fname"

recID	valueID
...	...
38	22
39	24
40	25
41	23
42	24
43	26
...	...

Dictionary "lname"

valueID	value
...	...
17	Brown
18	Doe
19	Miller
20	Schmidt
21	Smith
...	...

Attribute Vector "lname"

recID	valueID
...	...
38	19
39	21
40	17
41	18
42	18
43	20
...	...

Database Operations
DELETE - example

- Remove Jane Doe from the database table

Dictionary "fname"

valueID	value
...	...
22	Andrew
23	Jane
24	John
25	Mary
26	Peter
...	...

Attribute Vector "fname"

recID	valueID
...	...
38	22
39	24
40	25
41	23
42	24
43	26
...	...

Dictionary "lname"

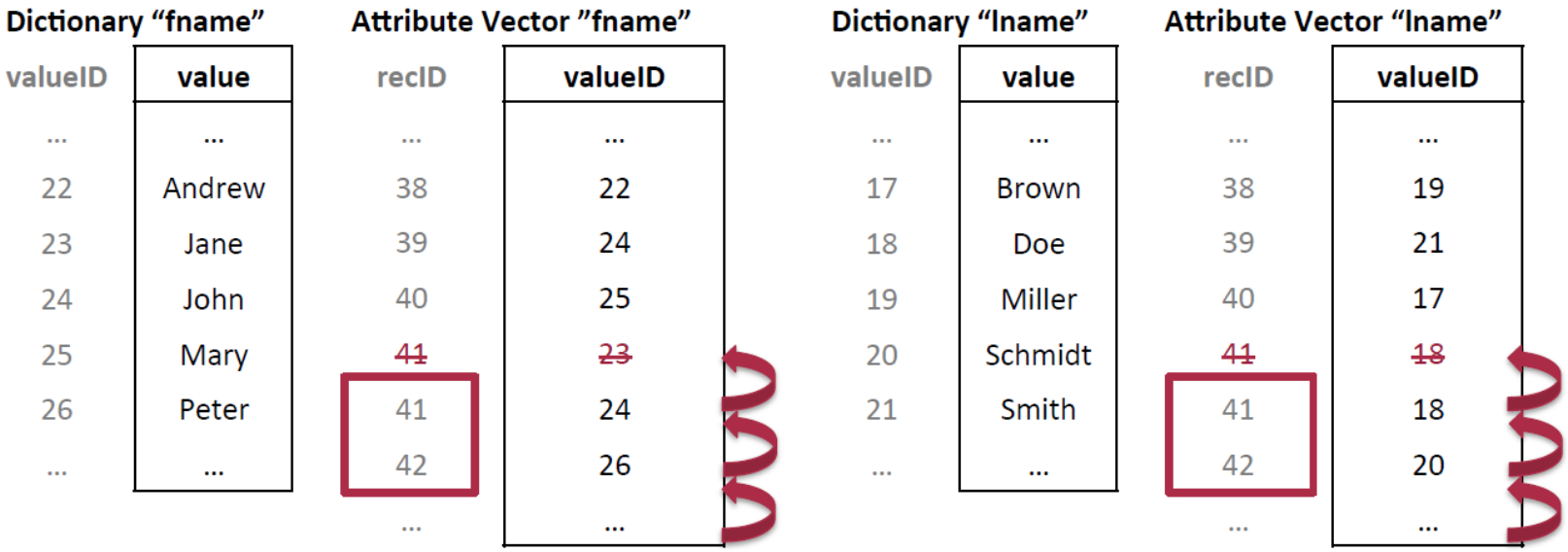
valueID	value
...	...
17	Brown
18	Doe
19	Miller
20	Schmidt
21	Smith
...	...

Attribute Vector "lname"

recID	valueID
...	...
38	19
39	21
40	17
41	18
42	18
43	20
...	...

Database Operations
DELETE - example

- Remove Jane Doe from the database table



INSERT

Database Operations
INSERT

- INSERT without new dictionary entry
  – New entry is already in dictionary, new valueID is appended to the attribute vector

- INSERT with new dictionary entry
  – New entry is added to the dictionary, dictionary is sorted, valueIDs are updated in attribute vector, new valueID is appended to the attribute vector

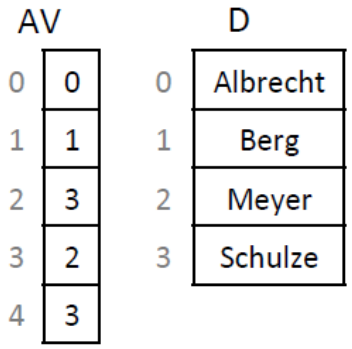
- SQL-Syntax:

```
INSERT INTO table_name
VALUES (value1,value2)
```

Database Operations

INSERT – example (Without New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
...	...	...	...	...	...	...

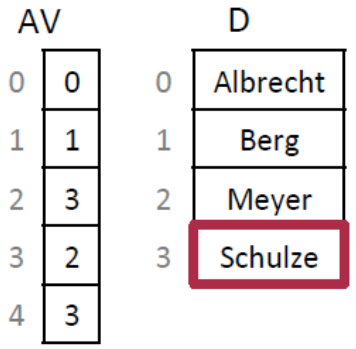
AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (Without New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
...	...	...	...	...	...	...

1. Look-up on dictionary → entry found

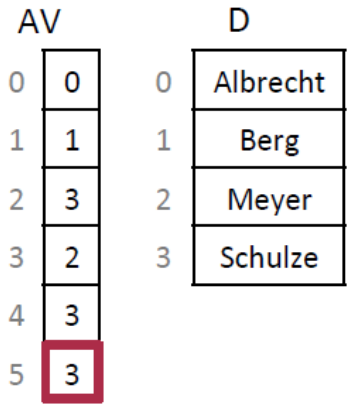
AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (Without New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5		Schulze				
...	...	...	...	...	...	...

1. Look-up on dictionary → entry found
2. Append valueID to attribute vector

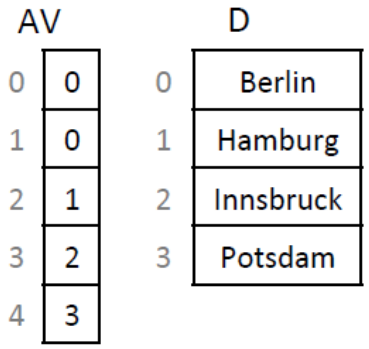
AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (With New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5		Schulze				
...	...	...	...	...	...	...

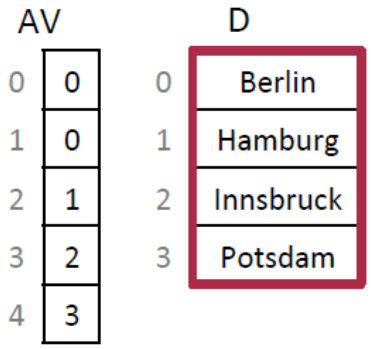
AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (With New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5		Schulze				
...	...	...	...	...	...	...

1. Look-up on dictionary → no entry found

AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (With New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)

	AV
0	0
1	0
2	1
3	2
4	3

	D
0	Berlin
1	Hamburg
2	Innsbruck
3	Potsdam
4	Rostock

	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5		Schulze				
...	...	...	...	...	...	...

1. Look-up on dictionary → no entry found
2. Append new value to dictionary

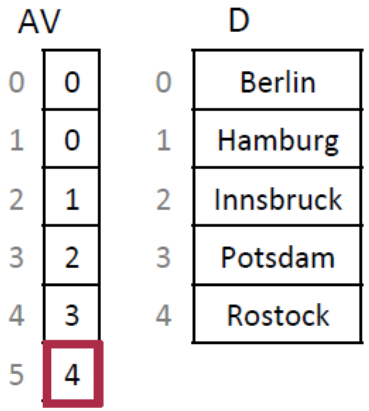
AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (With New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5		Schulze			Rostock	
...	...	...	...	...	...	...

1. Look-up on dictionary → no entry found
2. Append new value to dictionary
3. Append valueID to attribute vector

AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (With New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)

	AV
0	2
1	3
2	1
3	0
4	4

	D
0	Anton
1	Hanna
2	Martin
3	Michael
4	Sophie

	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5		Schulze			Rostock	
...	...	...	...	...	...	...

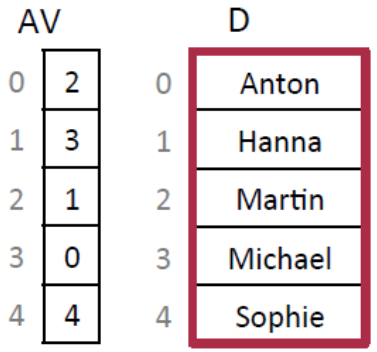
AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (With New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5		Schulze			Rostock	
...	...	...	...	...	...	...

1. Look-up on dictionary → no entry found

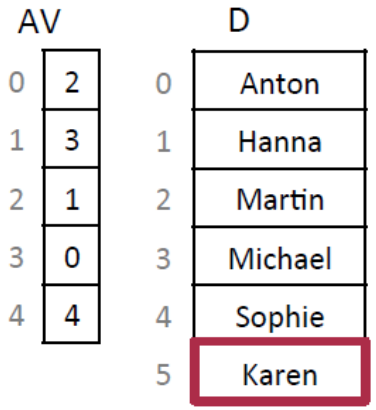
AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (With New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5		Schulze			Rostock	
...	...	...	...	...	...	...

1. Look-up on dictionary → no entry found
2. Append new value to dictionary

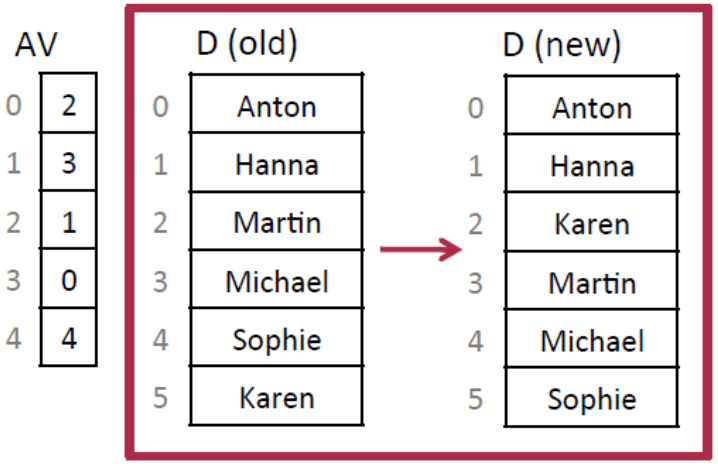
AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (With New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5		Schulze			Rostock	
...	...	...	...	...	...	...

1. Look-up on dictionary → no entry found
2. Append new value to dictionary
3. Sort Dictionary

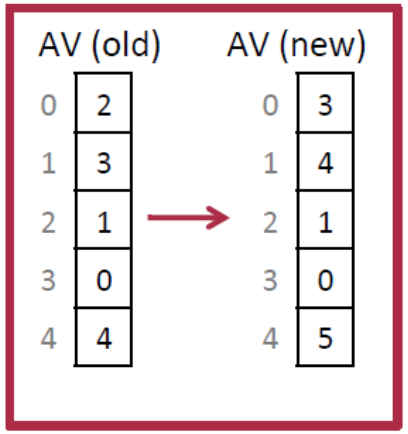
AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (With New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	D (new)
0	Anton
1	Hanna
2	Karen
3	Martin
4	Michael
5	Sophie

	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5		Schulze			Rostock	
...	...	...	...	...	...	...

1. Look-up on dictionary → no entry found
2. Append new value to dictionary
3. Sort Dictionary
4. Change valueIDs in attribute vector

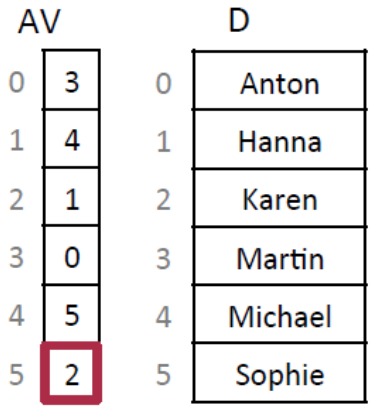
AV – Attribute Vector

D – Dictionary

Database Operations

INSERT – example (With New Dictionary Entry)

INSERT INTO world_population **VALUES** (Karen, Schulze, f, GER, Rostock, 06-20-2014)



	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Sophie	Schulze	f	GER	Potsdam	09-03-1977
5	Karen	Schulze			Rostock	
...	...	...	...	...	...	...

1. Look-up on dictionary → no entry found
2. Append new value to dictionary
3. Sort Dictionary
4. Change valueIDs in attribute vector
5. Append new valueID to attribute vector

AV – Attribute Vector

D – Dictionary

UPDATE

Database Operations
UPDATE

- Combination of DELETE and INSERT operation

- SQL-Syntax:

```
UPDATE world_population
SET city = „Bamberg“
WHERE fname = „Hanna“ AND lname = „Schulze“
```

recID	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Potsdam	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Ulrike	Schulze	f	GER	Potsdam	09-03-1977
5	Sophie	Schulze	f	GER	Rostock	06-20-2012
...	...	...	...	...	...	...
8×10^9	Zacharias	Perdopolus	m	GRE	Athen	03-12-1979

Database Operations
UPDATE – example

UPDATE world_population SET city = „Bamberg“ WHERE lname = „Schulze“

Dictionary old

1	Berlin
2	Hamburg
3	Innsbruck
4	Potsdam
5	Rostock

recID	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Ulrike	Schulze	f	GER	Potsdam	09-03-1977
5	Sophie	Schulze	f	GER	Rostock	06-20-2012
6	...	...	...	...	...	...

1. Look-up „Bamberg“ in dictionary → entry not found

Database Operations
UPDATE – example

UPDATE world_population SET city = „Bamberg“ WHERE lname = „Schulze“

Dictionary

	old
1	Berlin
2	Hamburg
3	Innsbruck
4	Potsdam
5	Rostock

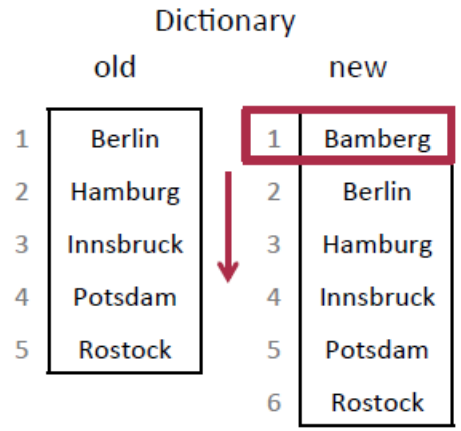
	new
1	Berlin
2	Hamburg
3	Innsbruck
4	Potsdam
5	Rostock
6	Bamberg

recID	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Ulrike	Schulze	f	GER	Potsdam	09-03-1977
5	Sophie	Schulze	f	GER	Rostock	06-20-2012
6	...	...	...	...	...	...

1. Look-up „Bamberg“ in dictionary → entry not found
2. Append new value „Bamberg“ to dictionary

Database Operations
UPDATE – example

UPDATE world_population SET city = „Bamberg“ WHERE lname = „Schulze“

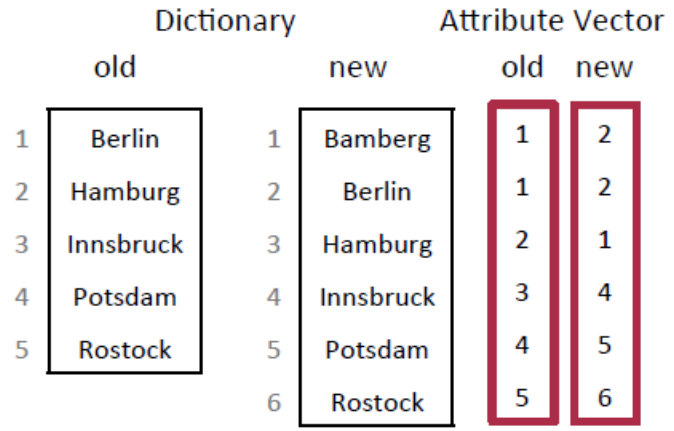


recID	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Ulrike	Schulze	f	GER	Potsdam	09-03-1977
5	Sophie	Schulze	f	GER	Rostock	06-20-2012
6	...	...	...	...	...	...

1. Look-up „Bamberg“ in dictionary → entry not found
2. Append new value „Bamberg“ to dictionary
3. Reorganize dictionary

Database Operations
UPDATE – example

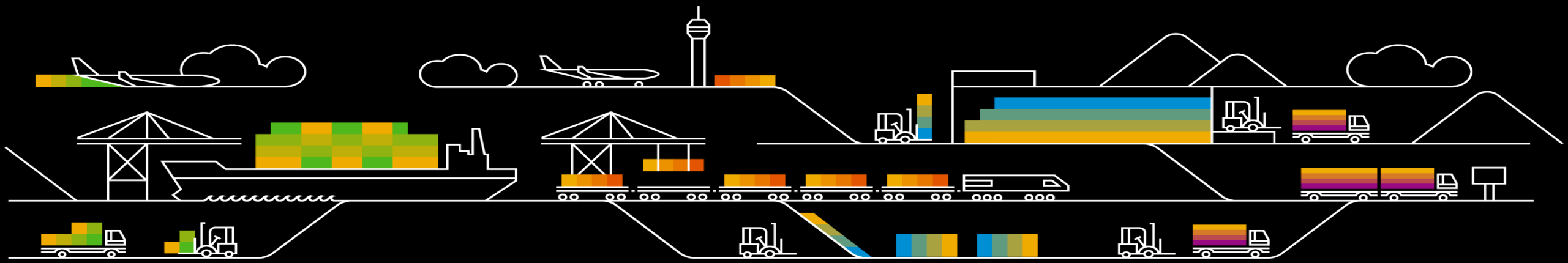
UPDATE world_population SET city = „Bamberg“ WHERE lname = „Schulze“



recID	fname	lname	gender	country	city	birthday
0	Martin	Albrecht	m	GER	Berlin	08-05-1955
1	Michael	Berg	m	GER	Berlin	03-05-1970
2	Hanna	Schulze	f	GER	Bamberg	04-04-1968
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992
4	Ulrike	Schulze	f	GER	Potsdam	09-03-1977
5	Sophie	Schulze	f	GER	Rostock	06-20-2012
6	...	...	...	...	...	...

1. Look-up „Bamberg“ in dictionary → entry not found
2. Append new value „Bamberg“ to dictionary
3. Reorganize dictionary
4. Replace old values with new values in attribute vector (expensive)

Demo



Demo
Performance

- System QM0 – 48 TB / 1100 CPUs

Table	Store	Rows	Size	Time
ACDOCA_C	Column	110 million	5 GB	

- System HANA Express edition (VM) – 16 GB / 4 CPUs

Table	Store	Rows	Size	Time

Demo
Performance

- System QM0 – 48 TB / 1100 CPUs

Table	Store	Rows	Size	Time
ACDOCA_C	Column	110 million	5 GB	1,8 s

- System HANA Express edition (VM) – 16 GB / 4 CPUs

Table	Store	Rows	Size	Time

Demo
Performance

- System QM0 – 48 TB / 1100 CPUs

Table	Store	Rows	Size	Time
ACDOCA_C	Column	110 million	5 GB	1,8 s
ACDOCA_R	Row	110 million	240 GB	

- System HANA Express edition (VM) – 16 GB / 4 CPUs

Table	Store	Rows	Size	Time

Demo
Performance

- System QM0 – 48 TB / 1100 CPUs

Table	Store	Rows	Size	Time
ACDOCA_C	Column	110 million	5 GB	1,8 s
ACDOCA_R	Row	110 million	240 GB	22,5 s

- System HANA Express edition (VM) – 16 GB / 4 CPUs

Table	Store	Rows	Size	Time

Demo
Performance

- System QM0 – 48 TB / 1100 CPUs

Table	Store	Rows	Size	Time
ACDOCA_C	Column	110 million	5 GB	1,8 s
ACDOCA_R	Row	110 million	240 GB	22,5 s
ACDOCA	Column	**19,5 billion**	1,3 TB	

- System HANA Express edition (VM) – 16 GB / 4 CPUs

Table	Store	Rows	Size	Time

Demo
Performance

- System QM0 – 48 TB / 1100 CPUs

Table	Store	Rows	Size	Time
ACDOCA_C	Column	110 million	5 GB	1,8 s
ACDOCA_R	Row	110 million	240 GB	22,5 s
ACDOCA	Column	**19,5 billion**	1,3 TB	139 s

- System HANA Express edition (VM) – 16 GB / 4 CPUs

Table	Store	Rows	Size	Time

Demo
Performance

- System QM0 – 48 TB / 1100 CPUs

Table	Store	Rows	Size	Time
ACDOCA_C	Column	110 million	5 GB	1,8 s
ACDOCA_R	Row	110 million	240 GB	22,5 s
ACDOCA	Column	**19,5 billion**	1,3 TB	139 s
ACDOCA_sm	Column	5 million	140 MB	0,5 s
CDHR	Column	31 million	1,3 GB	12,4 s
CDPOS	Column	730 million	44 GB	

- System HANA Express edition (VM) – 16 GB / 4 CPUs

Table	Store	Rows	Size	Time
ACDOCA_sm	Column	5 million	140 MB	0,9 s

Demo

Columns compression

Table Name: ACDOCA_C | Schema: I019855 | Type: Column Store

Columns | Indexes | Further Properties | Runtime Information

General

Total Memory Consumption (KB):	4 823 606	Memory Consumption in Main Storage (KB):	4 822 027
Number of Entries:	112 385 556	Memory Consumption in Delta Storage (KB):	1 579
Size on Disk (KB):	3 409 352	Estimated Maximum Memory Consumption (KB):	4 830 717

Details for Table

Parts | Columns

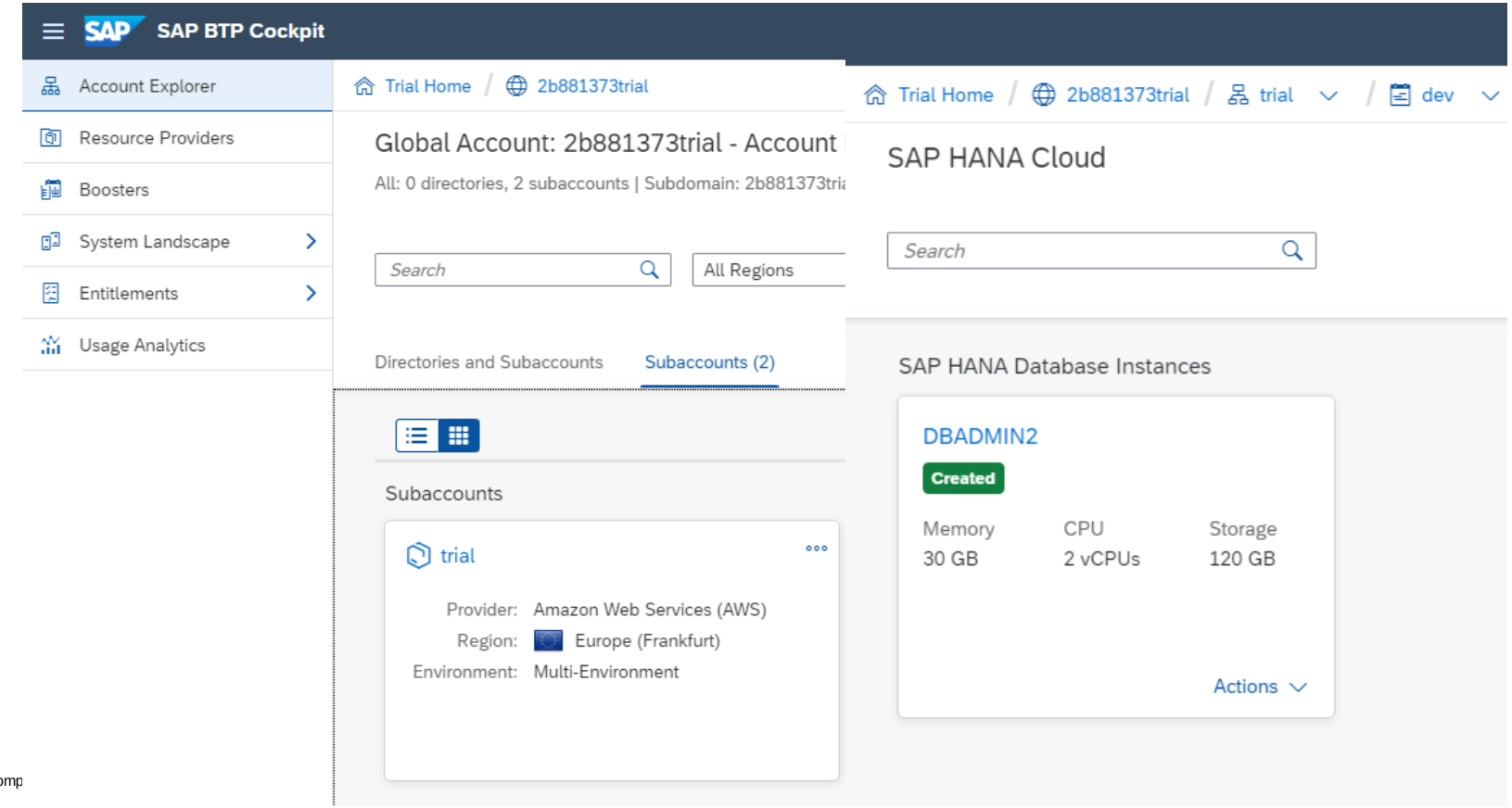
Column Name	Part ID	Host	Port	Total Size (KB)	Main Size (KB)	Delta Size (KB)	Main Size Compression Ratio [%]	Record Count	Distinct Records	Loaded	Compression Type
AWREF	0	lddbqm0	30203	963 953	963 949	4	80	112 385 556	30 125 780	TRUE	SPARSE
BELNR	0	lddbqm0	30203	892 284	892 280	4	74	112 385 556	22 536 153	TRUE	SPARSE
TIMESTAMP	0	lddbqm0	30203	287 527	287 523	4	33	112 385 556	3 826 735	TRUE	INDIRECT
HSL	0	lddbqm0	30203	206 997	206 993	4	24	112 385 556	6 925 294	TRUE	INDIRECT
TSL	0	lddbqm0	30203	206 739	206 735	4	24	112 385 556	6 924 248	TRUE	INDIRECT
WSL	0	lddbqm0	30203	205 254	205 250	4	23	112 385 556	6 886 661	TRUE	INDIRECT
KSL	0	lddbqm0	30203	200 710	200 706	4	23	112 385 556	6 705 872	TRUE	INDIRECT
OBJNR	0	lddbqm0	30203	109 640	109 636	4	19	112 385 556	101 052	TRUE	INDIRECT
PAROB1	0	lddbqm0	30203	91 089	91 085	4	19	112 385 556	17 121	TRUE	INDIRECT
CO_BELNR	0	lddbqm0	30203	80 587	80 583	4	17	112 385 556	887 016	TRUE	INDIRECT
OSL	0	lddbqm0	30203	152 639	152 635	4	17	112 385 556	6 362 864	TRUE	INDIRECT
AUFNR	0	lddbqm0	30203	31 383	31 379	4	12	112 385 556	18 556	TRUE	INDIRECT
ACCAS	0	lddbqm0	30203	38 946	38 942	4	9	112 385 556	98 120	TRUE	INDIRECT
ZUONR	0	lddbqm0	30203	37 558	37 554	4	9	112 385 556	2 468 987	TRUE	INDIRECT
PAUFNR	0	lddbqm0	30203	22 158	22 154	4	9	112 385 556	12 528	TRUE	CLUSTERED
MATNR	0	lddbqm0	30203	17 969	17 965	4	9	112 385 556	34 358	TRUE	INDIRECT
GKONT	0	lddbqm0	30203	71 939	71 935	4	8	112 385 556	1 030 641	TRUE	INDIRECT
VTSTAMP	0	lddbqm0	30203	60 972	60 968	4	7	112 385 556	3 461 543	TRUE	SPARSE
PACCAS	0	lddbqm0	30203	25 367	25 363	4	7	112 385 556	16 086	TRUE	INDIRECT
PRCTR	0	lddbqm0	30203	34 038	34 034	4	7	112 385 556	2 224	TRUE	INDIRECT
BLDAT	0	lddbqm0	30203	57 319	57 315	4	6	112 385 556	5 717	TRUE	INDIRECT
RLDNR	0	lddbqm0	30203	18 799	18 795	4	6	112 385 556	53	TRUE	INDIRECT
LIFNR	0	lddbqm0	30203	9 712	9 708	4	6	112 385 556	1 732	TRUE	RLE
AUGBL	0	lddbqm0	30203	7 173	7 169	4	6	112 385 556	55 298	TRUE	RLE

SAP HANA, Express Edition

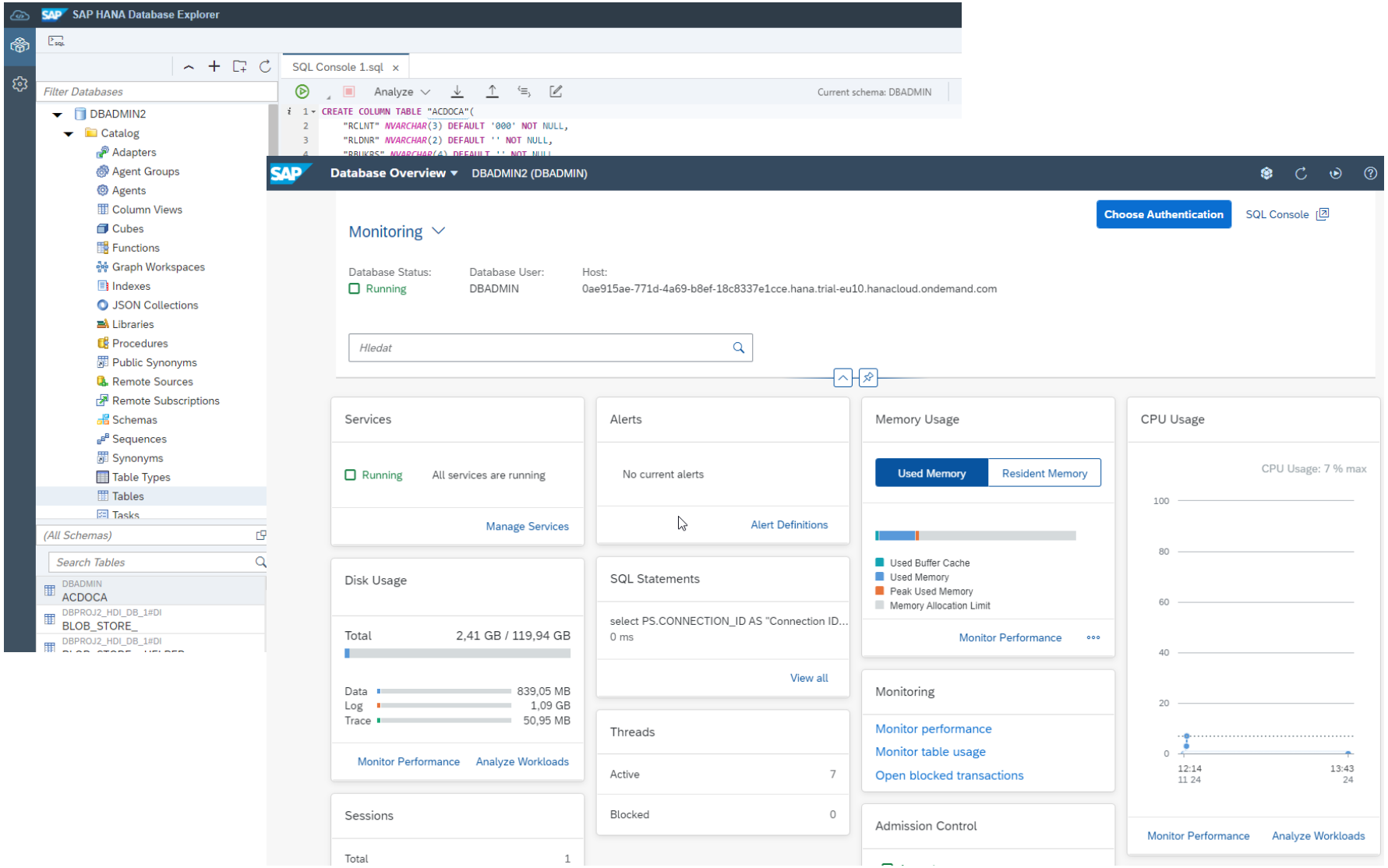
SAP HANA, express edition is a database and application development platform. You can run it for free (up to 32GB of RAM) on your laptop and start building new apps.

SAP HANA Cloud

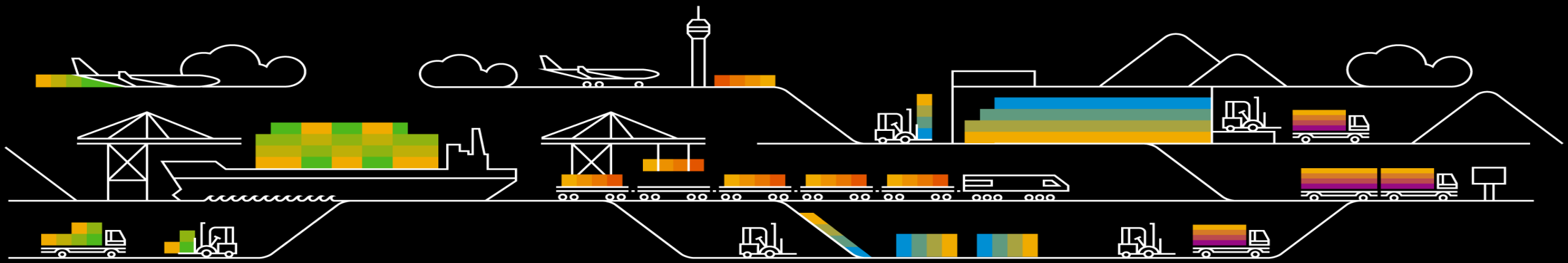
SAP HANA Cloud trial is a trial version of HANA DB. You can run it for free with following resources: 32GB of RAM, 120GB Storage, 2vCPU.



SAP HANA Cloud



Resources



Resources

- Plattner, Hasso. "In-Memory Data Management 2015" OpenHPI. Hasso-Plattner-Institute, 07 Sept. 2015. Web. 13 July 2017. https://open.hpi.de/courses/imdb2015
- SAP HANA Cloud https://developers.sap.com/topics/hana.html
- SAP HANA trial: https://www.sap.com/products/hana/express-trial.html
- SAP HANA Academy Videos: https://www.youtube.com/user/saphanaacademy
- SAP Help Portal - SAP HANA Platform: https://help.sap.com/viewer/product/SAP_HANA_PLATFORM/

Appendix

Thank you.

Radim Benek
Development Expert at SAP

Radim Benek
Development Expert, AIS Financials Brno, SAP Labs Czech Republic
SAP CR, spol. s r.o.
Holandská 2/4
639 00 Brno

radim.benek@sap.com
linkedin.com/in/radimbenek/

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

THE BEST RUN SAP