<epam>

Index Types

Relational Databases Basics

TRAINING
CENTER
<epam>

Just a quick reminder

Index – a specific kind of physical access path (an implementation construct, intended to improve the speed of access to data as physically stored).

Index for a database is like a map for a human. It helps finding objects of interest quickly and easy.

Disclaimer

This is just a short, brief, quick overview! “Under the hood” there is a lot of mathematics, algorithms and other stuff that may take years to comprehend.

Index types overview

By fields count	
Simple	Composite

By records uniqueness	
Unique	Non-unique

By records order	
Clustered	Non-clustered
Primary	

By storage	
Partitioned	Non-partitioned

By density	
Sparse	Dense

By hierarchy	
One-leveled	Multi-leveled

By correlation with query	
Covering	Non-covering

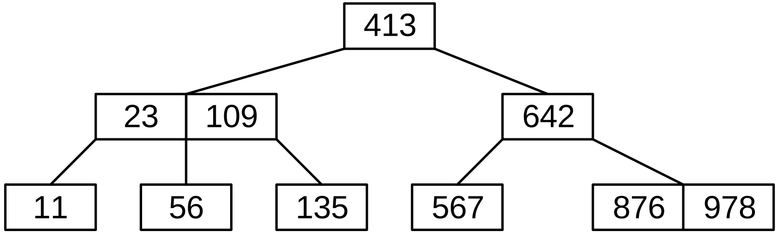
Index types overview

By basic structure
B-tree
T-tree
R-tree
Hash-table
Bit-mask
…

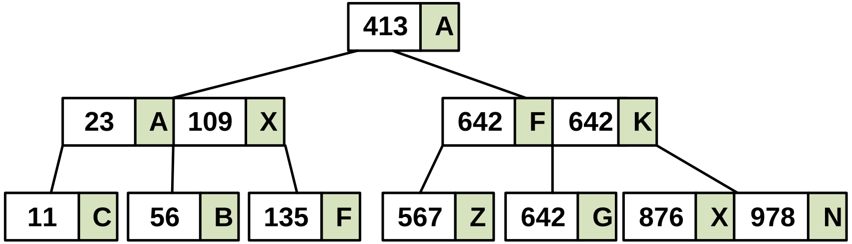
By specific functions
Column-store
With included columns
On computed columns
On function values
Filtered
Spatial
Full-text
Domain
XML
…

By fields count: simple and composite (compound)

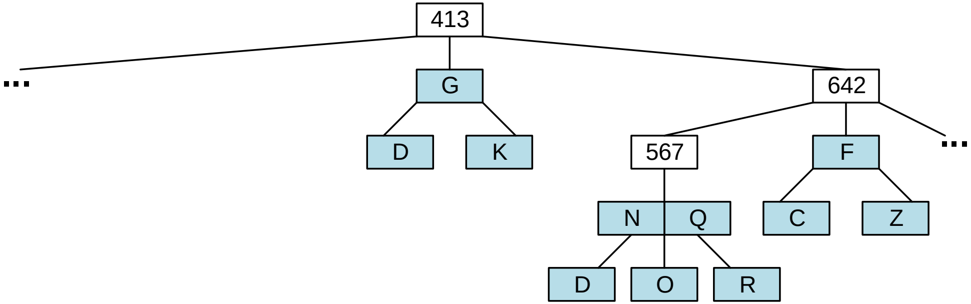
Simple (B-tree based)



Composite (B-tree based, key combinations)

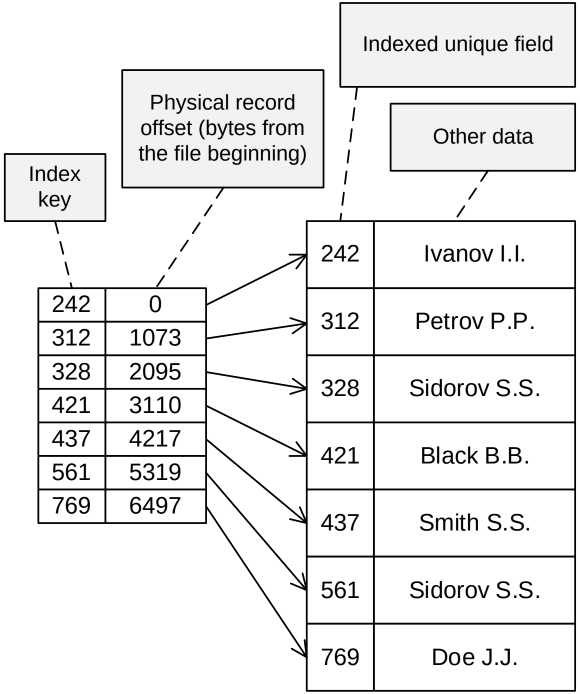


Composite (B-tree based, nested trees)



By records uniqueness: unique and non-unique

Unique



Index key

Physical record offset (bytes from the file beginning)

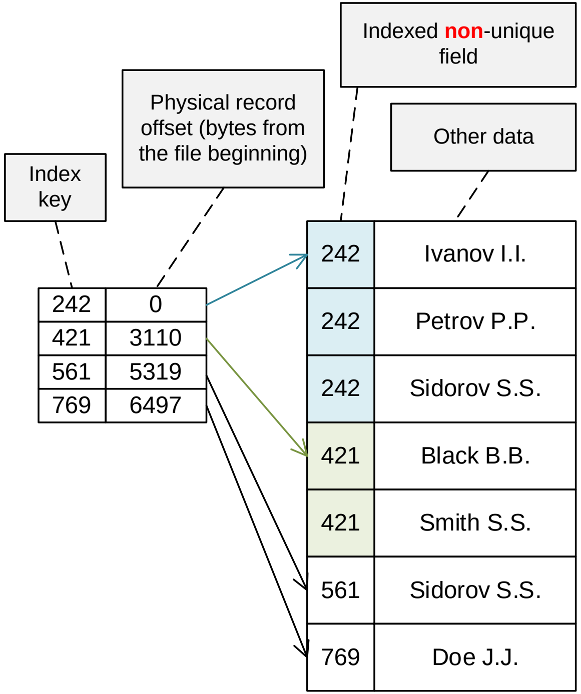
Indexed unique field

Other data

242	0
312	1073
328	2095
421	3110
437	4217
561	5319
769	6497

242	Ivanov I.I.
312	Petrov P.P.
328	Sidorov S.S.
421	Black B.B.
437	Smith S.S.
561	Sidorov S.S.
769	Doe J.J.

Non-unique



Index key

Physical record offset (bytes from the file beginning)

Indexed **non**-unique field

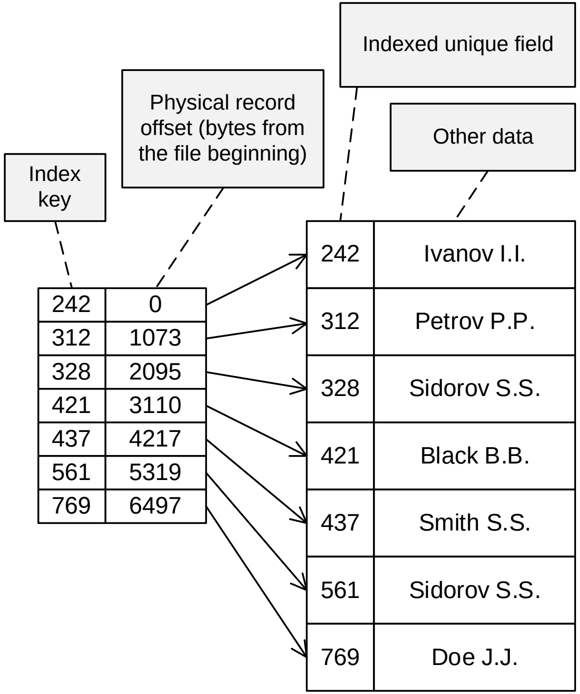
Other data

242	0
421	3110
561	5319
769	6497

242	Ivanov I.I.
242	Petrov P.P.
242	Sidorov S.S.
421	Black B.B.
421	Smith S.S.
561	Sidorov S.S.
769	Doe J.J.

By records order: primary and clustered

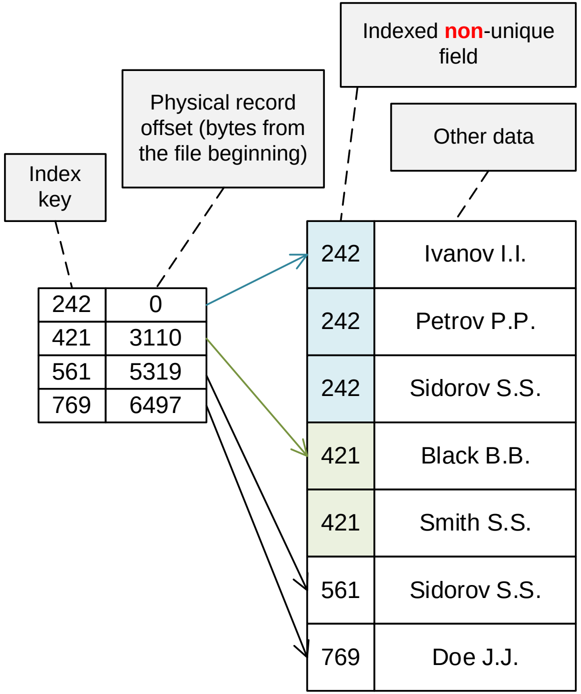
Primary



242	0
312	1073
328	2095
421	3110
437	4217
561	5319
769	6497

242	Ivanov I.I.
312	Petrov P.P.
328	Sidorov S.S.
421	Black B.B.
437	Smith S.S.
561	Sidorov S.S.
769	Doe J.J.

Clustered


Index key
Physical record offset (bytes from the file beginning)
Indexed **non**-unique field
Other data

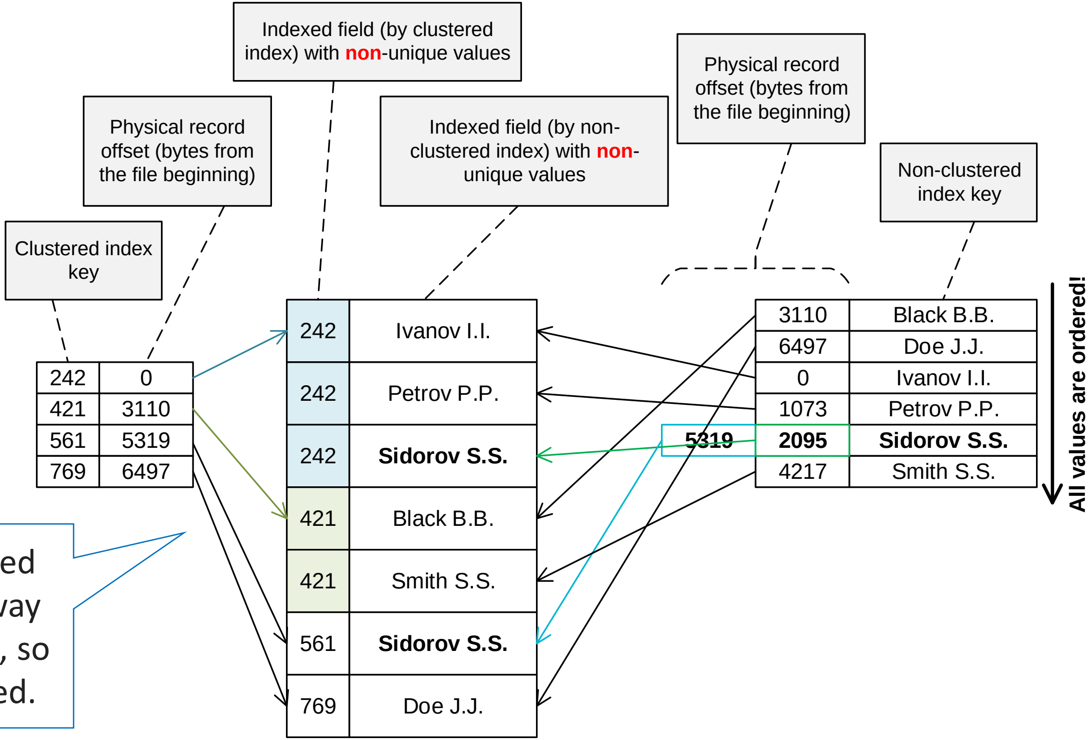
242	0
421	3110
561	5319
769	6497

242	Ivanov I.I.
242	Petrov P.P.
242	Sidorov S.S.
421	Black B.B.
421	Smith S.S.
561	Sidorov S.S.
769	Doe J.J.

By records order: clustered and non-clustered

Clustered

Non-clustered



Indexed field (by clustered index) with **non**-unique values
Physical record offset (bytes from the file beginning)
Indexed field (by non-clustered index) with **non**-unique values
Clustered index key
Physical record offset (bytes from the file beginning)
Non-clustered index key
All values are ordered!

242	0
421	3110
561	5319
769	6497

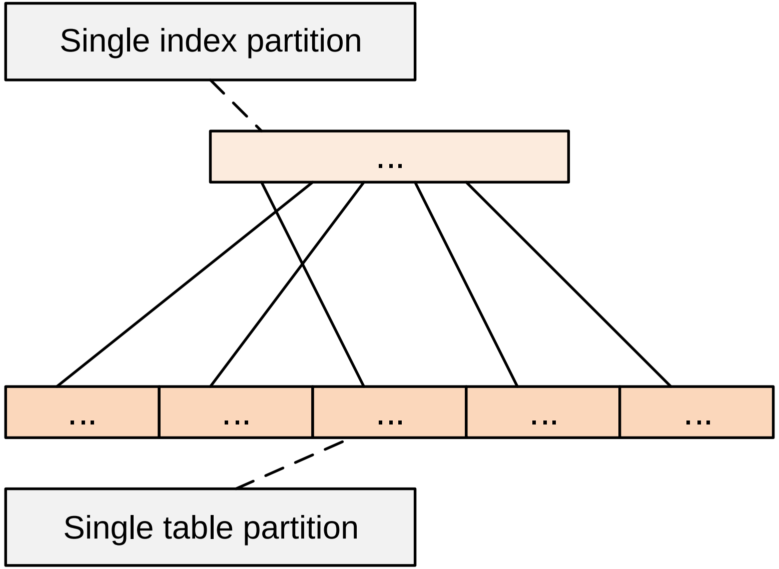
242	Ivanov I.I.
242	Petrov P.P.
242	**Sidorov S.S.**
421	Black B.B.
421	Smith S.S.
561	**Sidorov S.S.**
769	Doe J.J.

	3110	Black B.B.
	6497	Doe J.J.
	0	Ivanov I.I.
	1073	Petrov P.P.
5319	**2095**	**Sidorov S.S.**
	4217	Smith S.S.

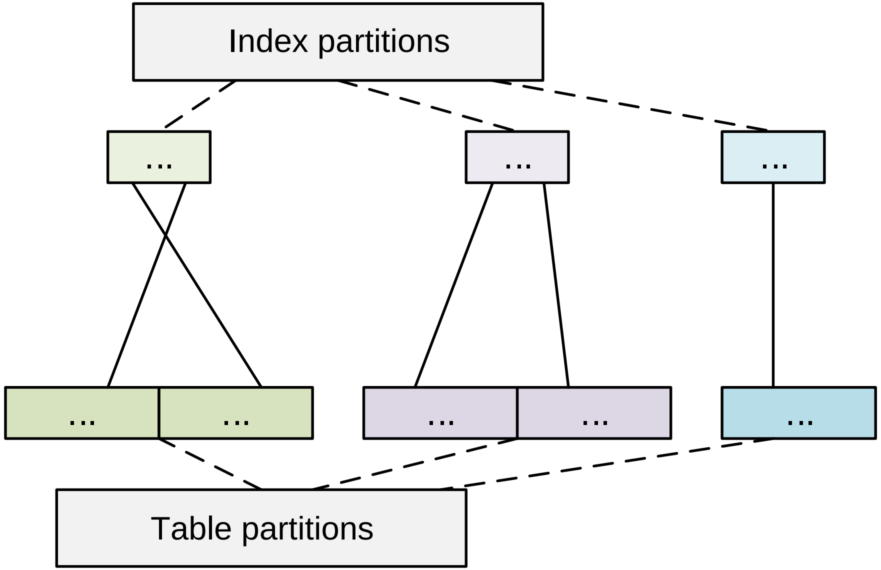
In most cases clustered indexes are just the way of table data ordering, so no data copies needed.

By storage: partitioned and non-partitioned

Non-partitioned table, non-partitioned index

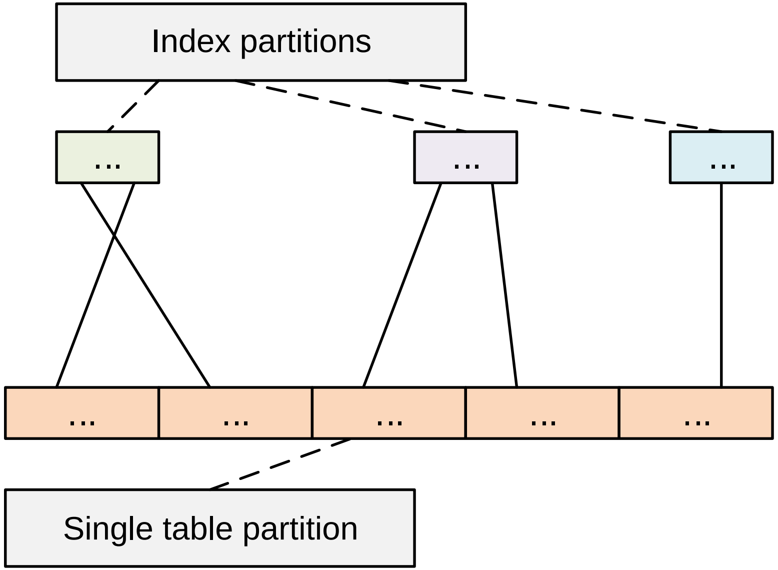


Partitioned table, partitioned index

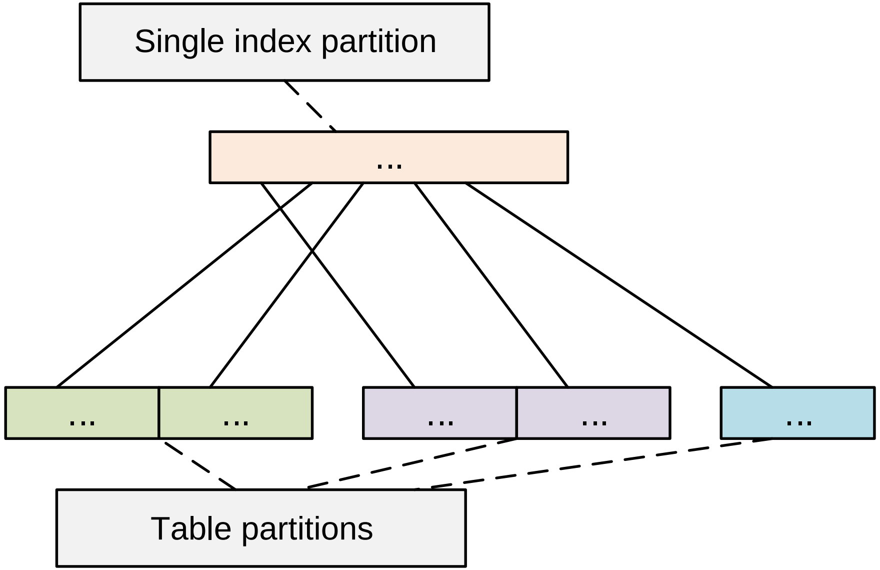


By storage: partitioned and non-partitioned

Non-partitioned table, partitioned index

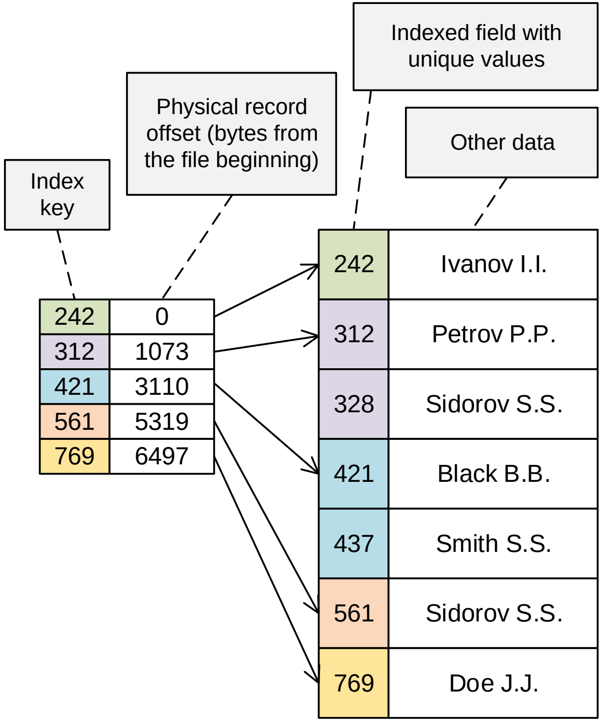


Partitioned table, non-partitioned index



By density: sparse and dense

Sparse



Index key

Physical record offset (bytes from the file beginning)

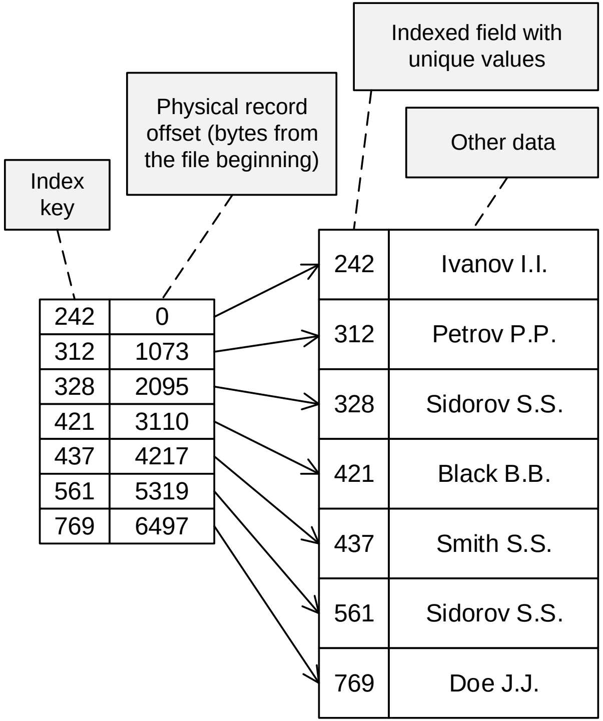
Indexed field with unique values

Other data

242	0
312	1073
421	3110
561	5319
769	6497

242	Ivanov I.I.
312	Petrov P.P.
328	Sidorov S.S.
421	Black B.B.
437	Smith S.S.
561	Sidorov S.S.
769	Doe J.J.

Dense



Index key

Physical record offset (bytes from the file beginning)

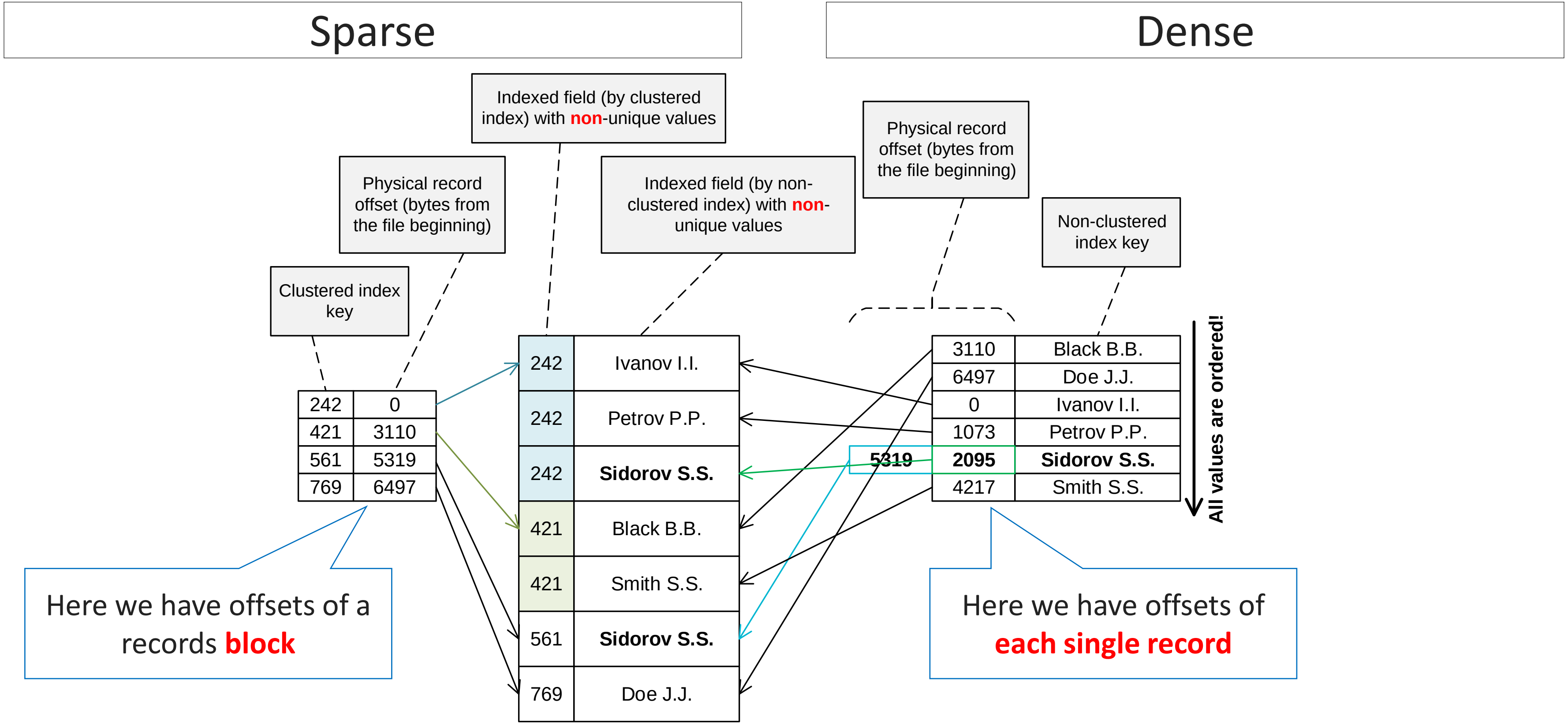
Indexed field with unique values

Other data

242	0
312	1073
328	2095
421	3110
437	4217
561	5319
769	6497

242	Ivanov I.I.
312	Petrov P.P.
328	Sidorov S.S.
421	Black B.B.
437	Smith S.S.
561	Sidorov S.S.
769	Doe J.J.

By density: sparse and dense



Sparse

Dense

Indexed field (by clustered index) with **non**-unique values

Physical record offset (bytes from the file beginning)

Indexed field (by non-clustered index) with **non**-unique values

Clustered index key

Physical record offset (bytes from the file beginning)

Non-clustered index key

242	0
421	3110
561	5319
769	6497

242	Ivanov I.I.
242	Petrov P.P.
242	**Sidorov S.S.**
421	Black B.B.
421	Smith S.S.
561	**Sidorov S.S.**
769	Doe J.J.

	3110	Black B.B.
	6497	Doe J.J.
	0	Ivanov I.I.
	1073	Petrov P.P.
~~**5319**~~	**2095**	**Sidorov S.S.**
	4217	Smith S.S.

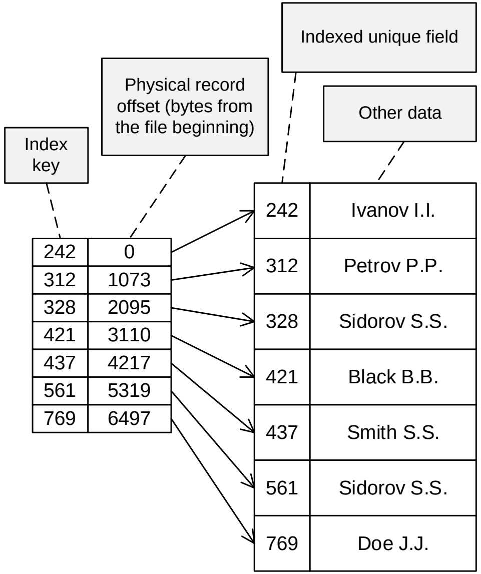
Here we have offsets of a records **block**

Here we have offsets of **each single record**

By hierarchy: one-leveled and multi-leveled

One-leveled	
Primary	Clustered
Hash-table based	Bit-mask based

Multi-leveled	
B-tree based	T-tree based
R-tree based	XML



Index key

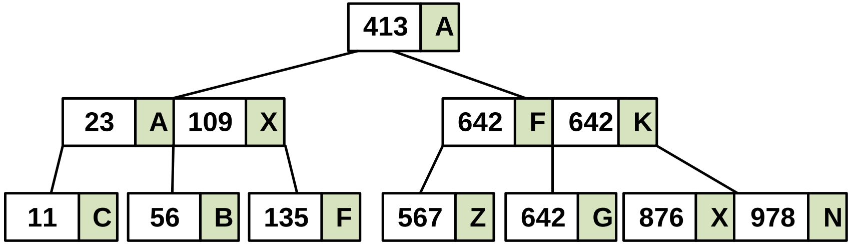
Physical record offset (bytes from the file beginning)

Indexed unique field

Other data

242	0
312	1073
328	2095
421	3110
437	4217
561	5319
769	6497

242	Ivanov I.I.
312	Petrov P.P.
328	Sidorov S.S.
421	Black B.B.
437	Smith S.S.
561	Sidorov S.S.
769	Doe J.J.



By correlation with query: covering and non-covering

Covering

Non-covering

Primary key, clustered index	Combined non-clustered index {u_email, u_status}		No indexes on these fields	
u_id	u_email	u_status	u_login	u_name
1	ivanov@mail.ru	Active	ivanov	Ivanov I.I.
2	petrov@mail.ru	Active	petrov	Petrov P.P.
3	sidorov@mail.ru	Locked	sidorov	Sidorov S.S.
4	smith@gmail.com	Active	smith	Smith S.S.
5	doe@yahoo.com	Locked	doe	Doe J.J.

```
SELECT COUNT(`u_id`)
FROM   `users`
WHERE  `u_id` >= 2 AND `u_id` <= 10
```

```
SELECT `u_status`
FROM   `users`
WHERE  `u_email` = 'ivanov@mail.ru'
```

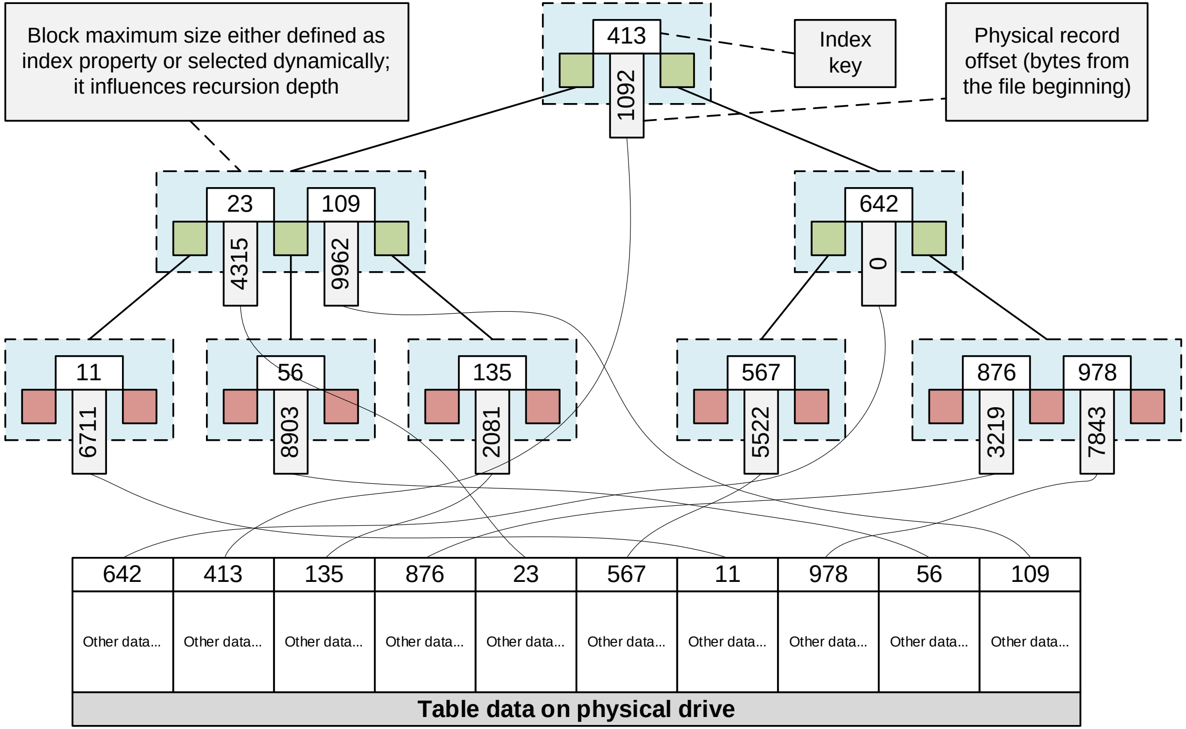
DBMS may retrieve all the necessary data for these queries directly from indexes

```
SELECT `u_login`
FROM   `users`
WHERE  `u_email` = 'ivanov@mail.ru'
   AND `u_status` = 'Active'
```

DBMS has to access table data (i.e., it can NOT retrieve all the necessary data for this query directly from index)

By basic structure: B-tree

B-tree with equal nodes (each node stores a record address)

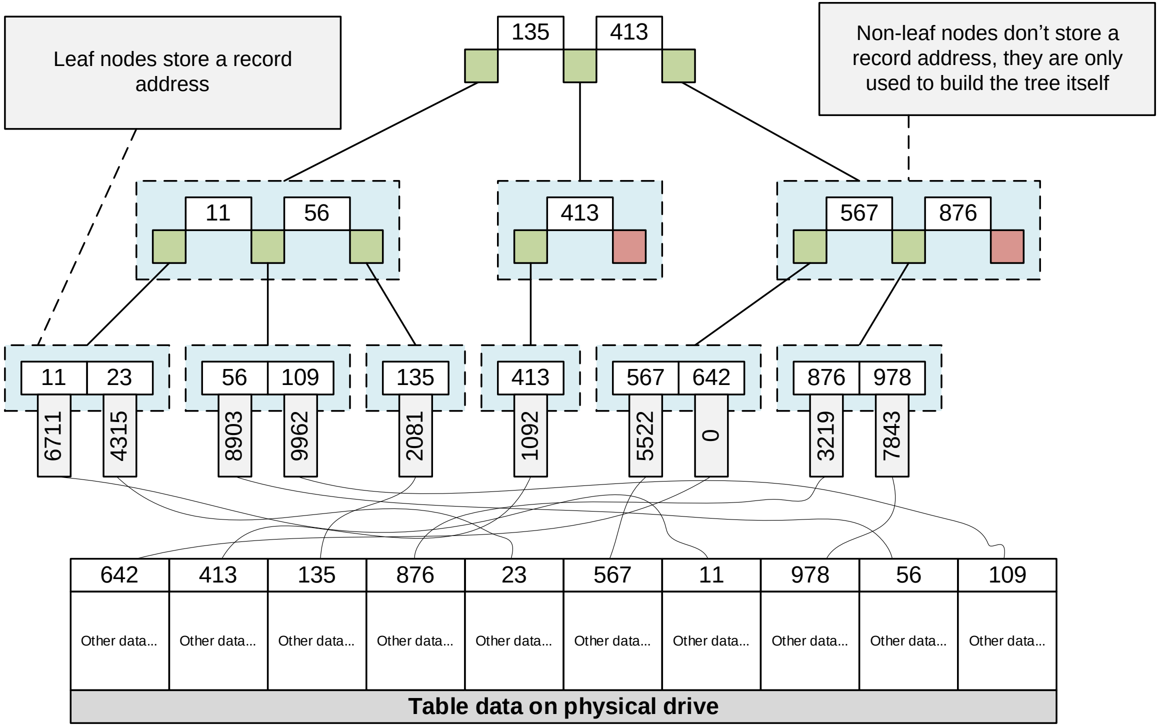

Block maximum size either defined as index property or selected dynamically; it influences recursion depth
413
1092
Index key
Physical record offset (bytes from the file beginning)
23
4315
109
9962
642
0
11
6711
56
8903
135
2081
567
5522
876
3219
978
7843

642	413	135	876	23	567	11	978	56	109
Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...

Table data on physical drive

By basic structure: B-tree

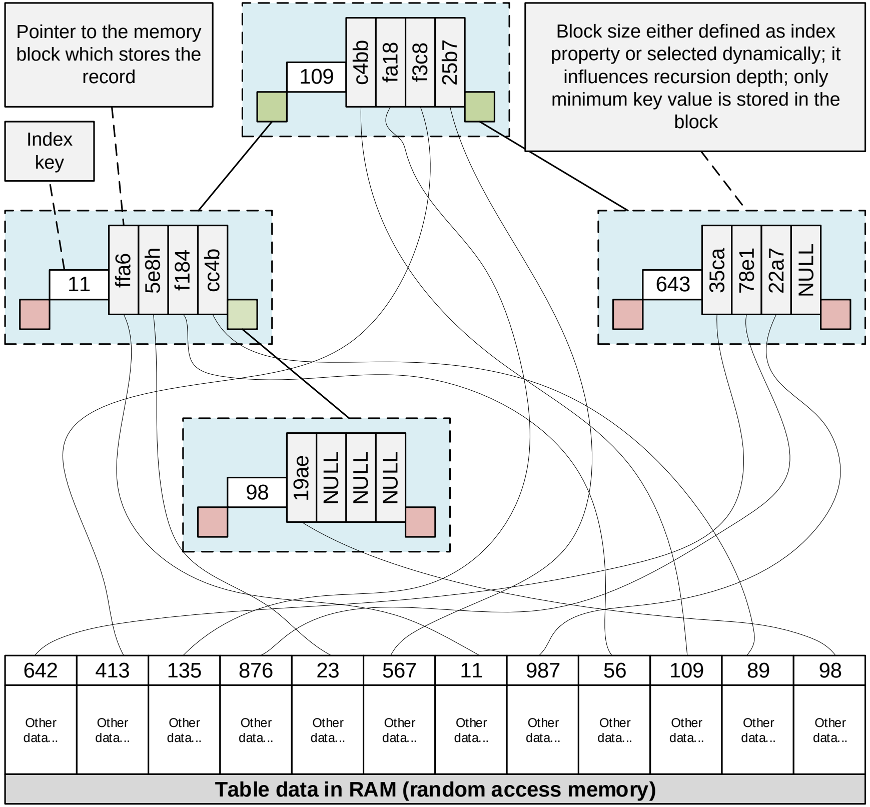
B-tree with non-equal nodes (only leaf nodes store a record address)



Leaf nodes store a record address
Non-leaf nodes don’t store a record address, they are only used to build the tree itself
135
413
11
56
413
567
876
11
23
56
109
135
413
567
642
876
978
6711
4315
8903
9962
2081
1092
5522
0
3219
7843
642
413
135
876
23
567
11
978
56
109
Other data...
Table data on physical drive

By basic structure: T-tree

T-tree



Pointer to the memory block which stores the record

Index key

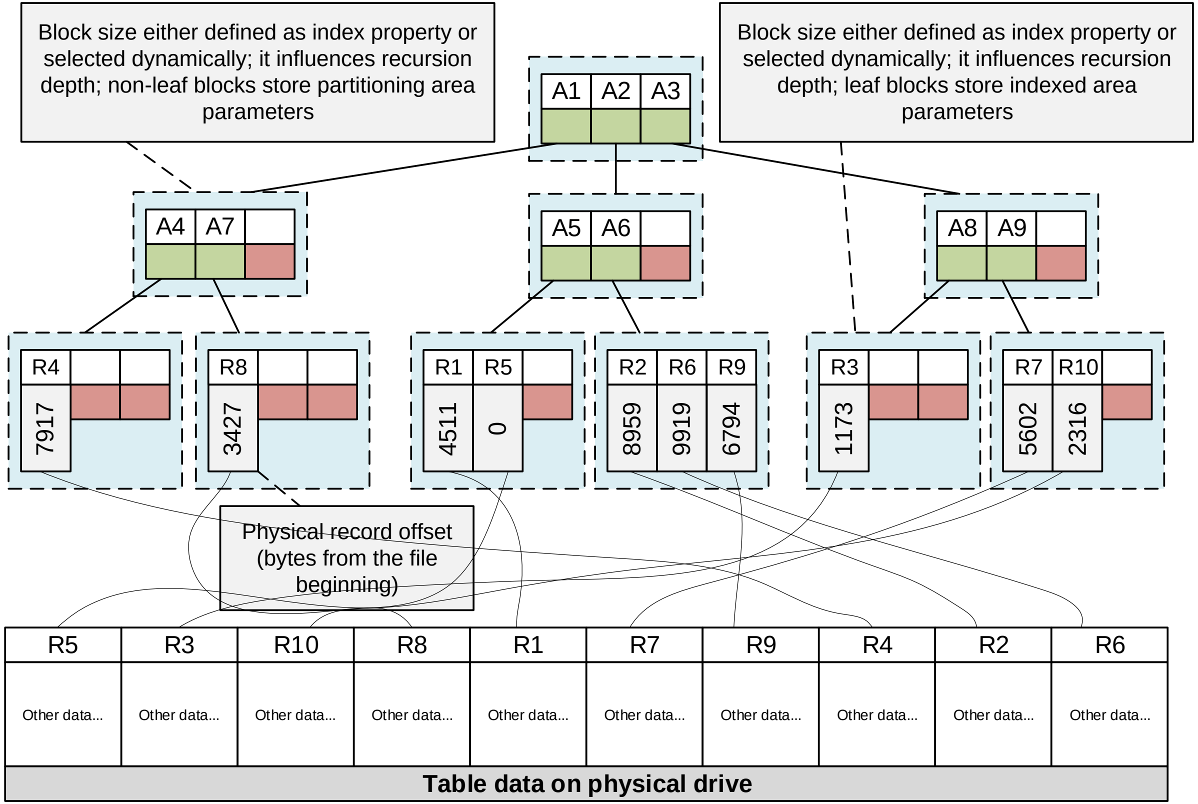
Block size either defined as index property or selected dynamically; it influences recursion depth; only minimum key value is stored in the block

642	413	135	876	23	567	11	987	56	109	89	98
Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...

Table data in RAM (random access memory)

By basic structure: R-tree

R-tree

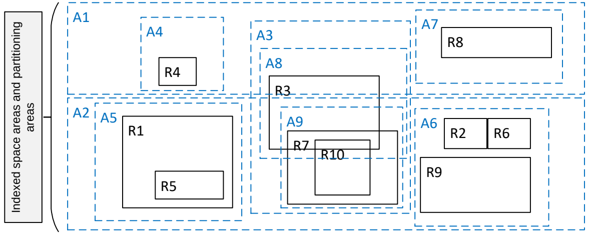

Block size either defined as index property or selected dynamically; it influences recursion depth; non-leaf blocks store partitioning area parameters

Block size either defined as index property or selected dynamically; it influences recursion depth; leaf blocks store indexed area parameters

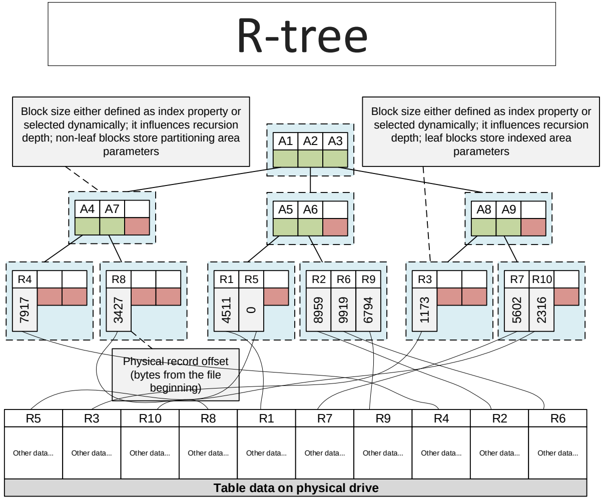
Physical record offset (bytes from the file beginning)

R5	R3	R10	R8	R1	R7	R9	R4	R2	R6
Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...

Table data on physical drive



By basic structure: R-tree



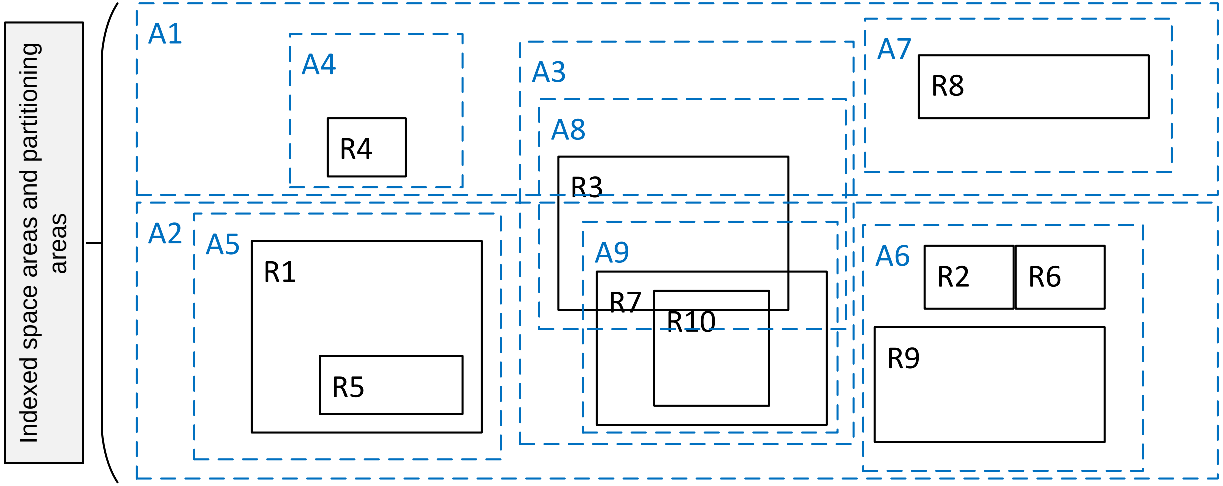
R-tree

Block size either defined as index property or selected dynamically; it influences recursion depth; non-leaf blocks store partitioning area parameters

Block size either defined as index property or selected dynamically; it influences recursion depth; leaf blocks store indexed area parameters

Physical record offset (bytes from the file beginning)

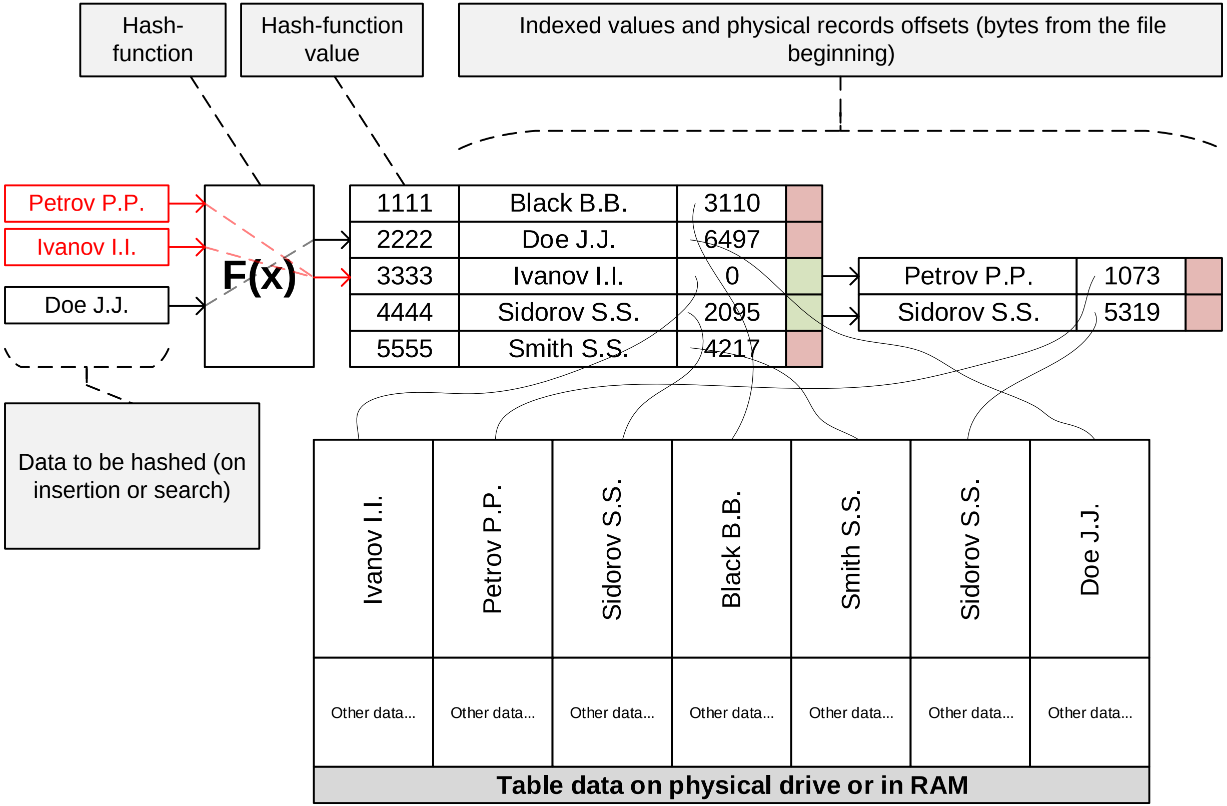
Table data on physical drive



Indexed space areas and partitioning areas

By basic structure: hash-table

Hash-table


Hash-function
Hash-function value
Indexed values and physical records offsets (bytes from the file beginning)
Petrov P.P.
Ivanov I.I.
Doe J.J.
F(x)
Data to be hashed (on insertion or search)

1111	Black B.B.	3110	🟥
2222	Doe J.J.	6497	🟥
3333	Ivanov I.I.	0	🟩
4444	Sidorov S.S.	2095	🟩
5555	Smith S.S.	4217	🟥

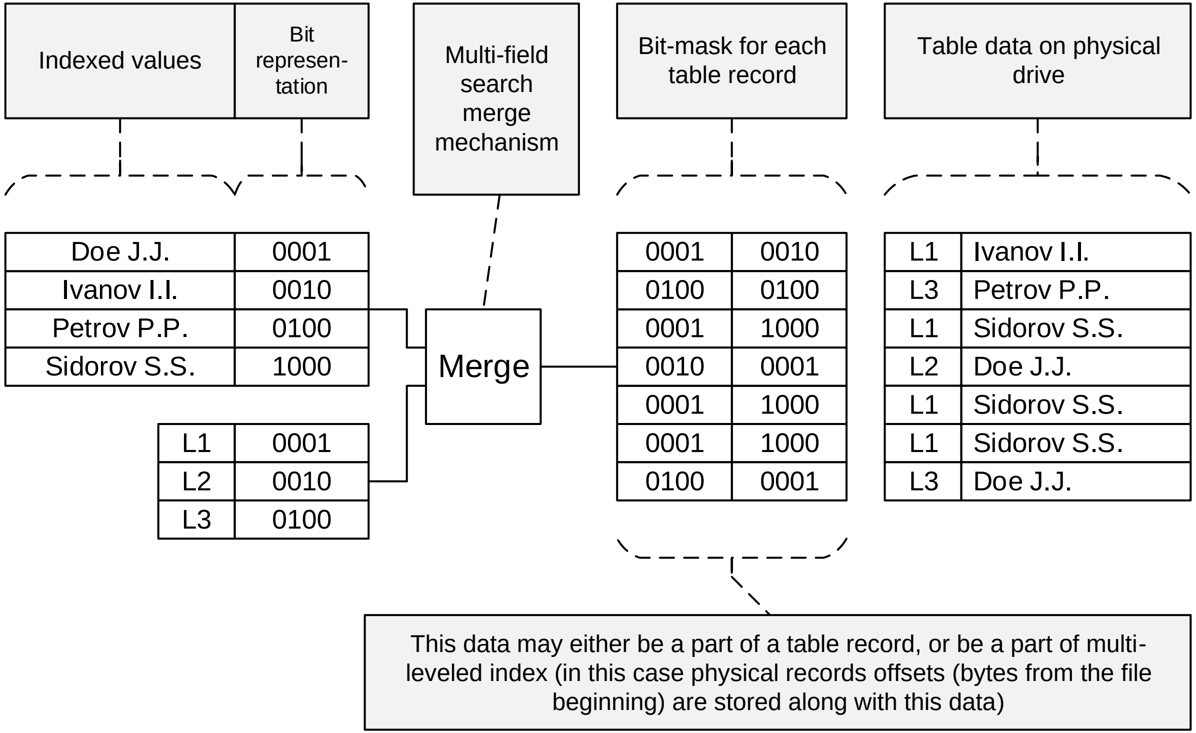
Petrov P.P.	1073	🟥
Sidorov S.S.	5319	🟥

Ivanov I.I.	Petrov P.P.	Sidorov S.S.	Black B.B.	Smith S.S.	Sidorov S.S.	Doe J.J.
Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...

Table data on physical drive or in RAM

By basic structure: bit-mask

Bit-mask



Indexed values	Bit represen-tation
Doe J.J.	0001
Ivanov I.I.	0010
Petrov P.P.	0100
Sidorov S.S.	1000

L1	0001
L2	0010
L3	0100

Bit-mask for each table record

0001	0010
0100	0100
0001	1000
0010	0001
0001	1000
0001	1000
0100	0001

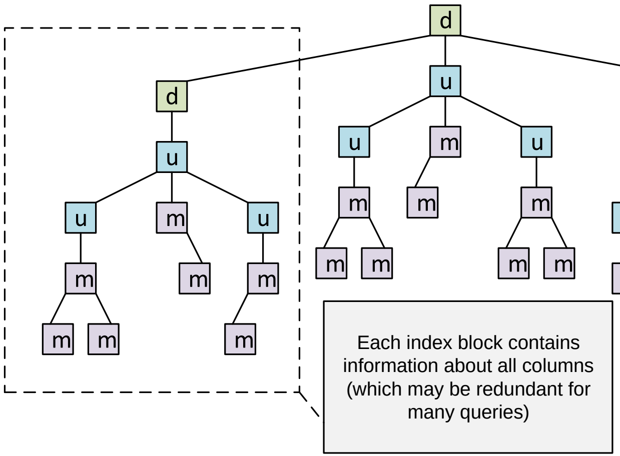
Table data on physical drive

L1	Ivanov I.I.
L3	Petrov P.P.
L1	Sidorov S.S.
L2	Doe J.J.
L1	Sidorov S.S.
L1	Sidorov S.S.
L3	Doe J.J.

This data may either be a part of a table record, or be a part of multi-leveled index (in this case physical records offsets (bytes from the file beginning) are stored along with this data)

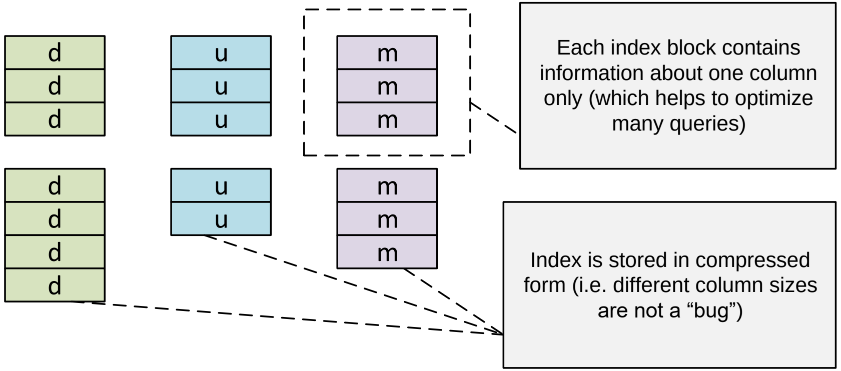
By specific functions: column-store

B-tree index



Each index block contains information about all columns (which may be redundant for many queries)

Column-store index



Each index block contains information about one column only (which helps to optimize many queries)

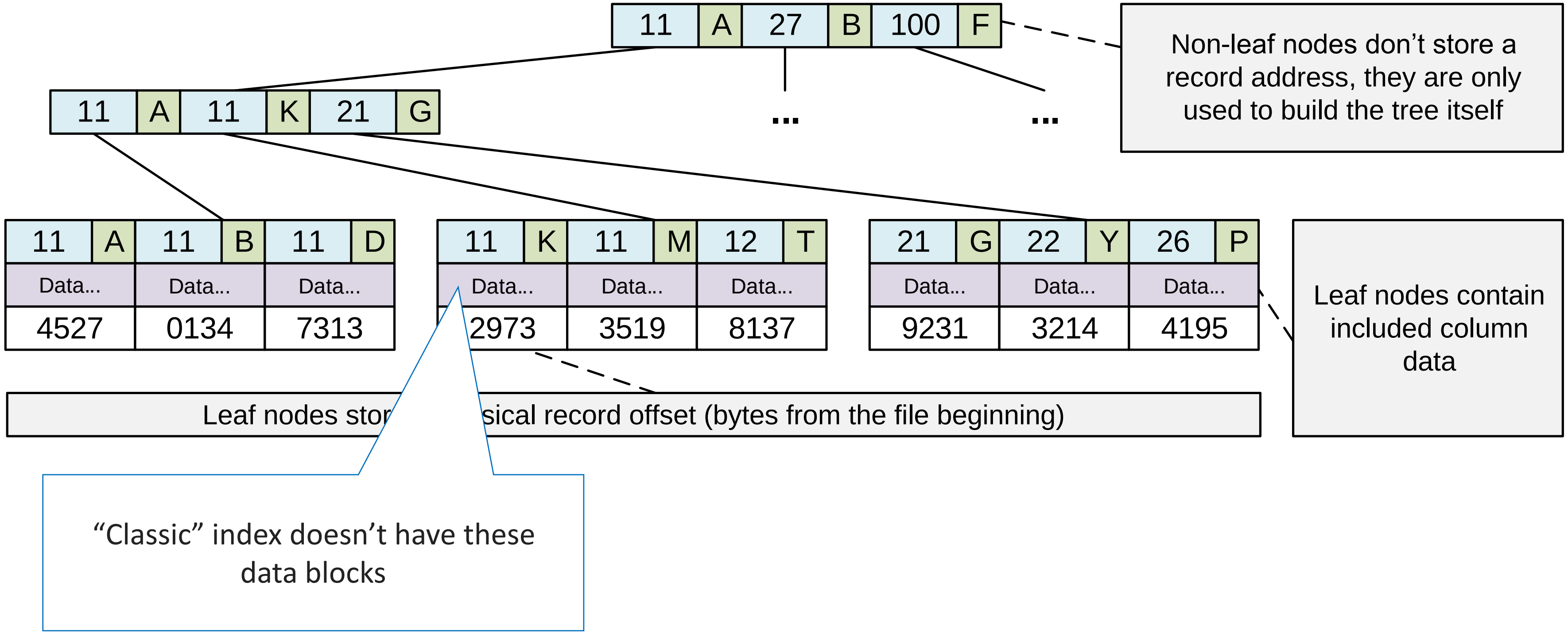
Index is stored in compressed form (i.e. different column sizes are not a “bug”)

Typical data sample. For such data column-store index may be efficient during a lot of analytical queries.

p_id	p_user	p_money	p_date	...
1	234	34556	2016-02-12	...
2	89	565	2016-03-18	...
3	34	341235	2015-09-02	...
4	2342	24234	2017-02-14	...
...	...	...	...	...
34526256	34235	21321	2016-12-19	...

By specific functions: with included columns

With included columns



Non-leaf nodes don’t store a record address, they are only used to build the tree itself

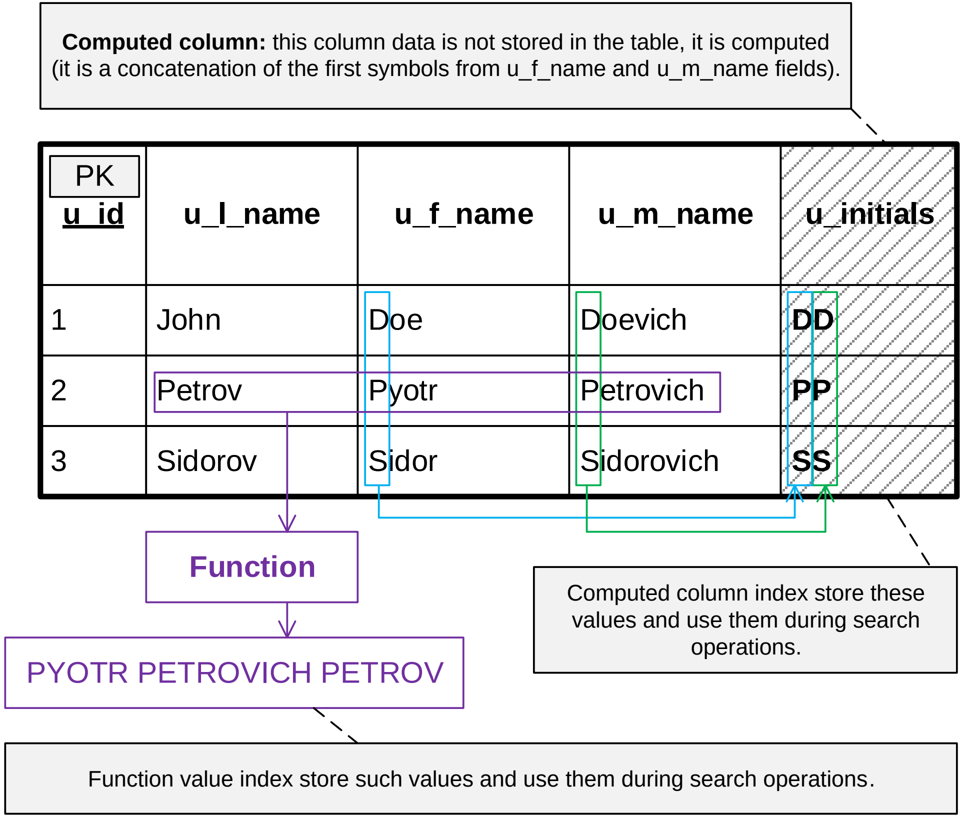
Leaf nodes contain included column data

Leaf nodes stor sical record offset (bytes from the file beginning)

“Classic” index doesn’t have these data blocks

By specific functions: on function values (or on computed columns)

On function values (or on computed columns)



Computed column: this column data is not stored in the table, it is computed (it is a concatenation of the first symbols from u_f_name and u_m_name fields).

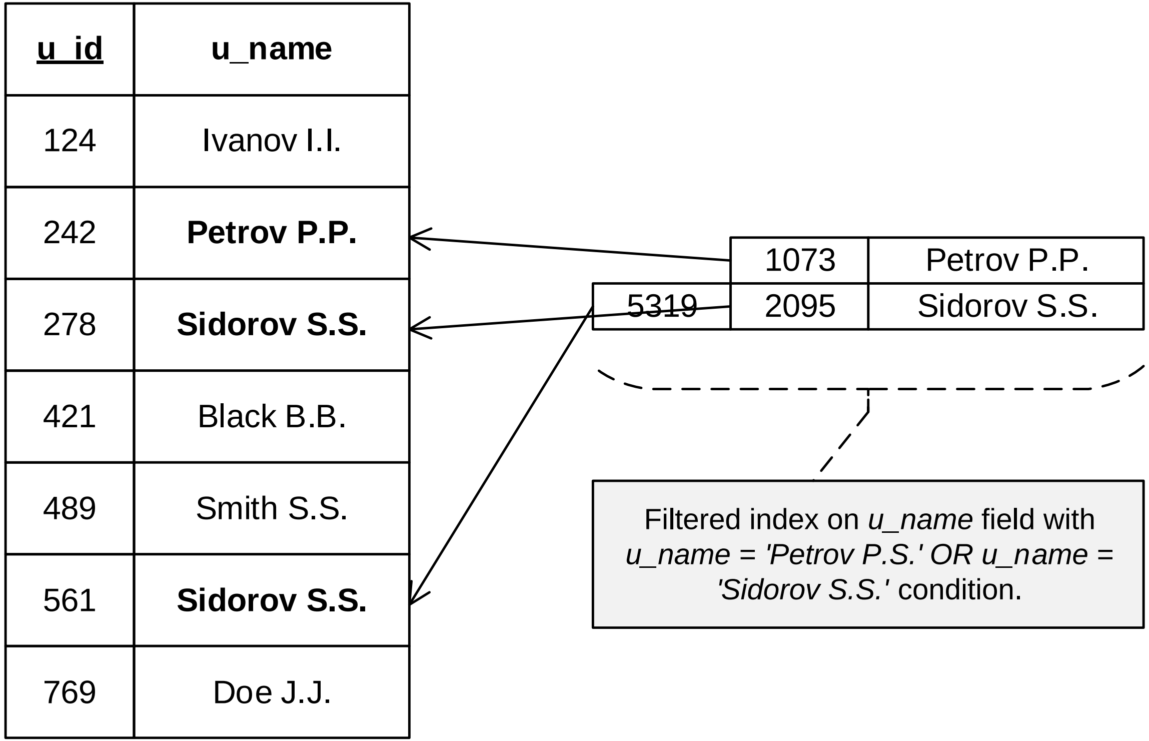
PK u_id	u_l_name	u_f_name	u_m_name	u_initials
1	John	Doe	Doevich	**DD**
2	Petrov	Pyotr	Petrovich	**PP**
3	Sidorov	Sidor	Sidorovich	**SS**

Computed column index store these values and use them during search operations.

Function value index store such values and use them during search operations.

By specific functions: filtered

Filtered



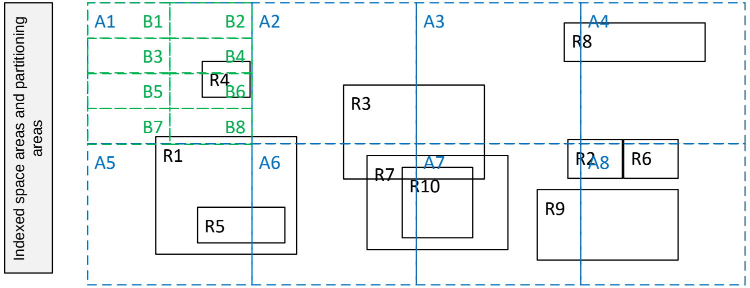
<u>u_id</u>	**u_name**
124	Ivanov I.I.
242	**Petrov P.P.**
278	**Sidorov S.S.**
421	Black B.B.
489	Smith S.S.
561	**Sidorov S.S.**
769	Doe J.J.

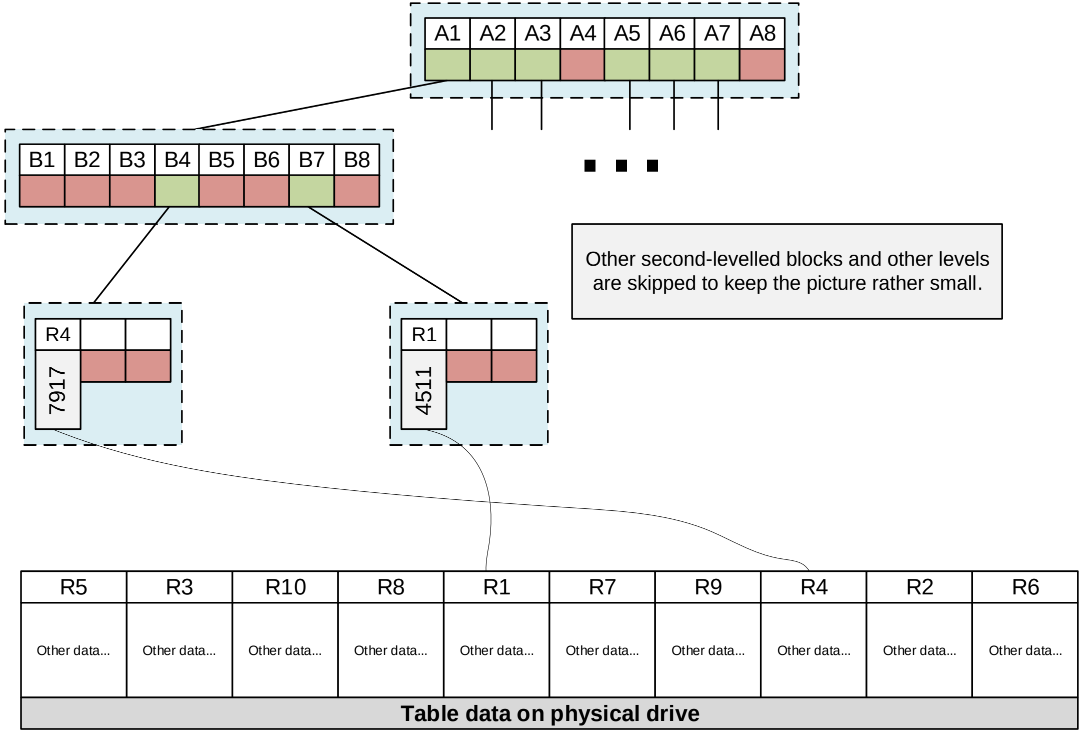
	1073	Petrov P.P.
5319	2095	Sidorov S.S.

Filtered index on *u_name* field with *u_name* = *'Petrov P.S.' OR u_name* = *'Sidorov S.S.'* condition.

By specific functions: spatial (usually, R-tree based)

Spatial (usually, R-tree based)





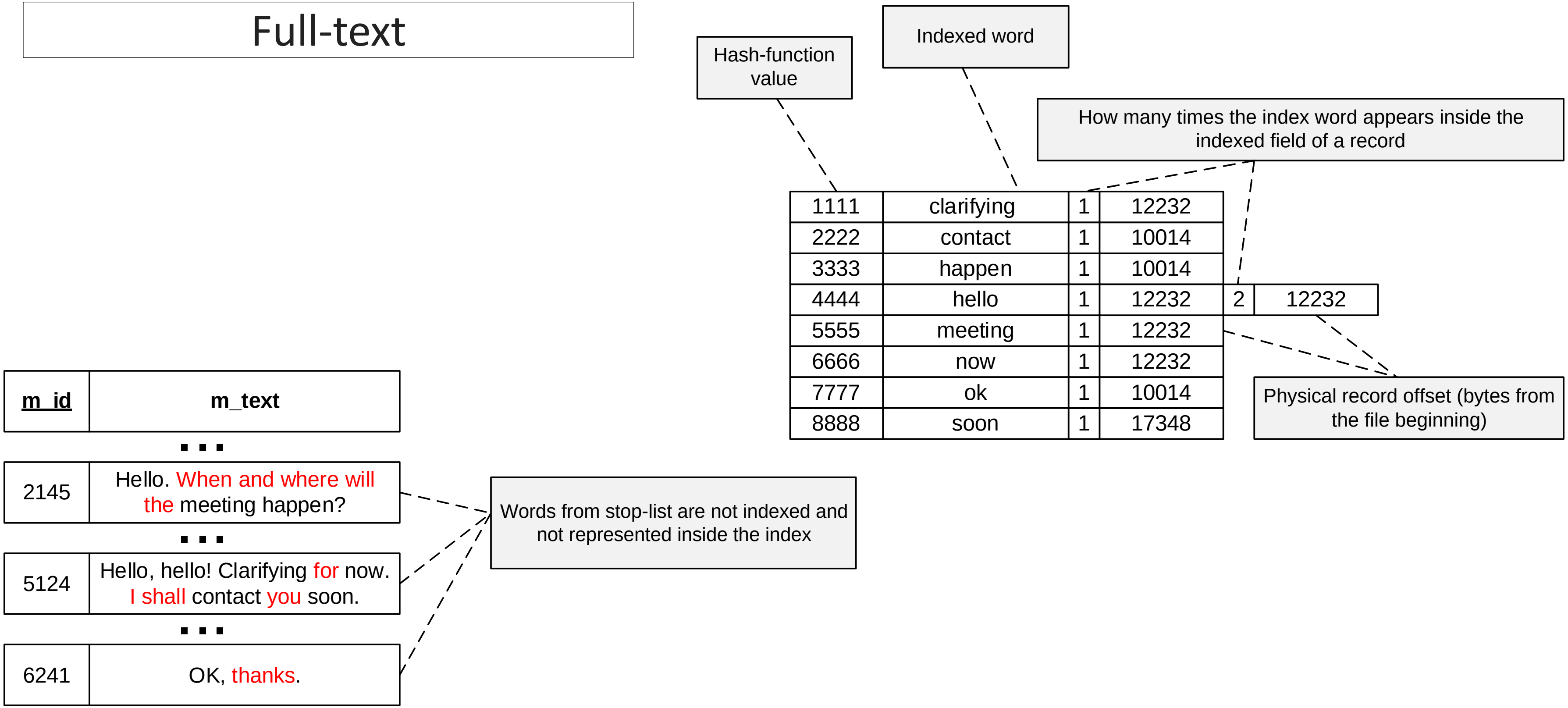
Other second-levelled blocks and other levels are skipped to keep the picture rather small.

R5	R3	R10	R8	R1	R7	R9	R4	R2	R6
Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...	Other data...

Table data on physical drive

By specific functions: full-text

Full-text



Hash-function value

Indexed word

How many times the index word appears inside the indexed field of a record

Physical record offset (bytes from the file beginning)

1111	clarifying	1	12232		
2222	contact	1	10014		
3333	happen	1	10014		
4444	hello	1	12232	2	12232
5555	meeting	1	12232		
6666	now	1	12232		
7777	ok	1	10014		
8888	soon	1	17348		

m_id	**m_text**
…	…
2145	Hello. When and where will the meeting happen?
…	…
5124	Hello, hello! Clarifying for now. I shall contact you soon.
…	…
6241	OK, thanks.

Words from stop-list are not indexed and not represented inside the index

By specific functions: domain

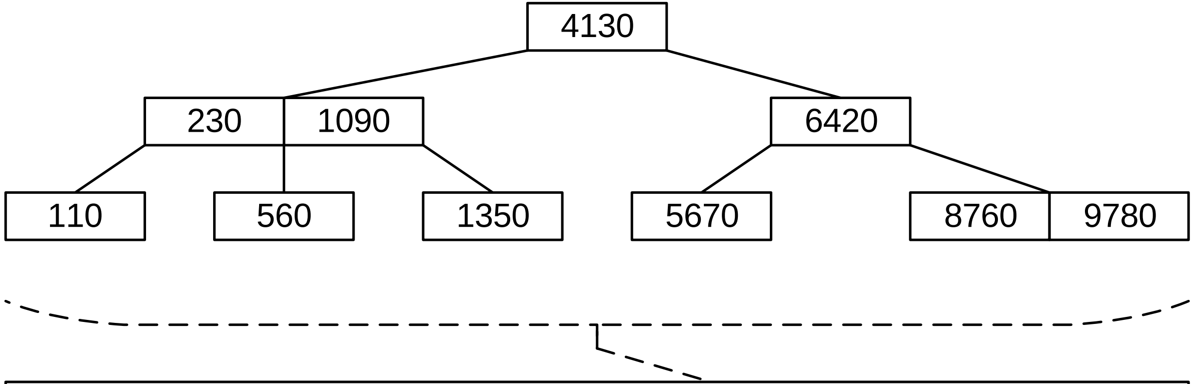
Domain

Full-text search with word form consideration

Time intervals processing (considering length and intersection)

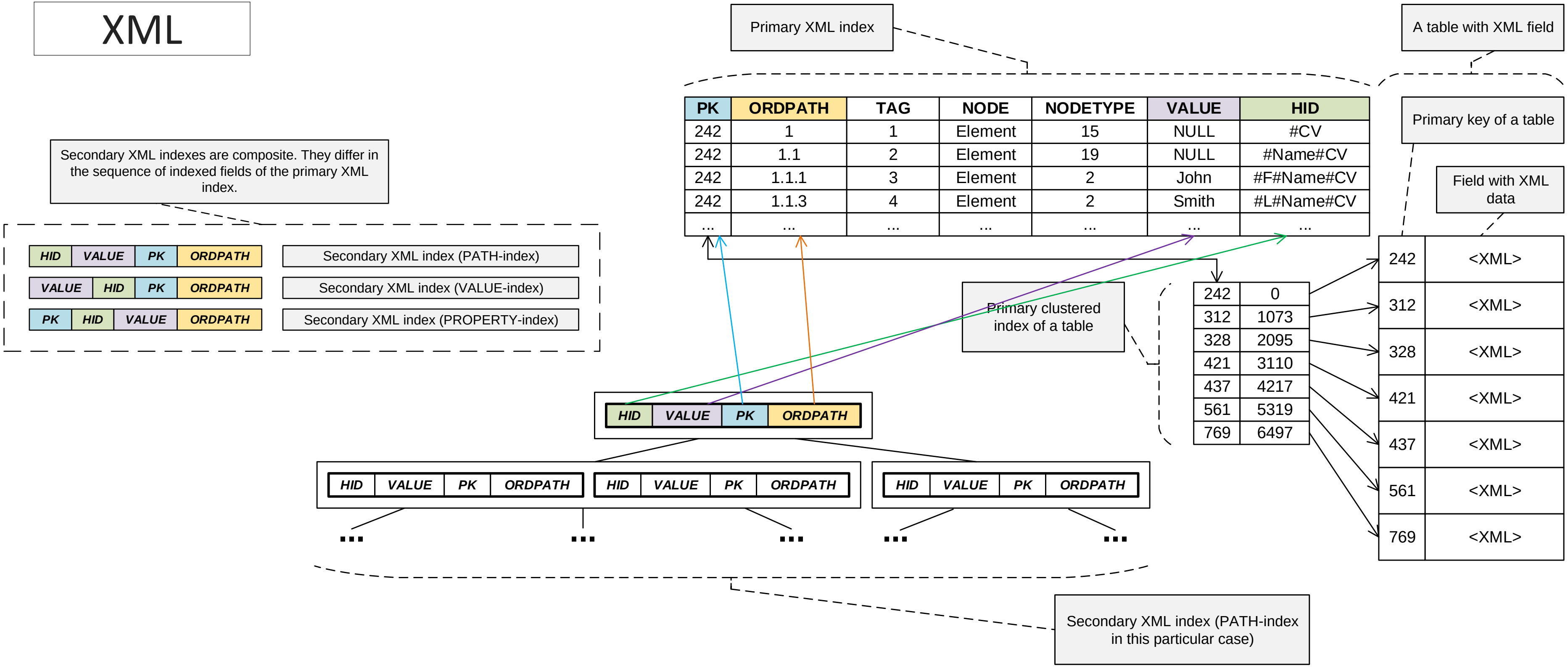
Serialized data indexing

Binary documents indexing (e.g., PDF)



Domain index may be B-tree based (on “any-other-structure-based)”. The main domain-index feature is special algorithms for such structure creation and usage.

By specific functions: XML



XML

Primary XML index

A table with XML field

PK	ORDPATH	TAG	NODE	NODETYPE	VALUE	HID
242	1	1	Element	15	NULL	#CV
242	1.1	2	Element	19	NULL	#Name#CV
242	1.1.1	3	Element	2	John	#F#Name#CV
242	1.1.3	4	Element	2	Smith	#L#Name#CV
...	...	...	...	...	...	...

Primary key of a table

Field with XML data

Secondary XML indexes are composite. They differ in the sequence of indexed fields of the primary XML index.

HID	VALUE	PK	ORDPATH	Secondary XML index (PATH-index)
VALUE	HID	PK	ORDPATH	Secondary XML index (VALUE-index)
PK	HID	VALUE	ORDPATH	Secondary XML index (PROPERTY-index)

Primary clustered index of a table

242	0
312	1073
328	2095
421	3110
437	4217
561	5319
769	6497

242	<XML>
312	<XML>
328	<XML>
421	<XML>
437	<XML>
561	<XML>
769	<XML>

HID | VALUE | PK | ORDPATH
HID | VALUE | PK | ORDPATH
HID | VALUE | PK | ORDPATH
HID | VALUE | PK | ORDPATH
...
...
...
...
...

Secondary XML index (PATH-index in this particular case)

And many, many others…

New index types appear almost every month…

Index Types

Relational Databases Basics