15.1 Multi-level indexes

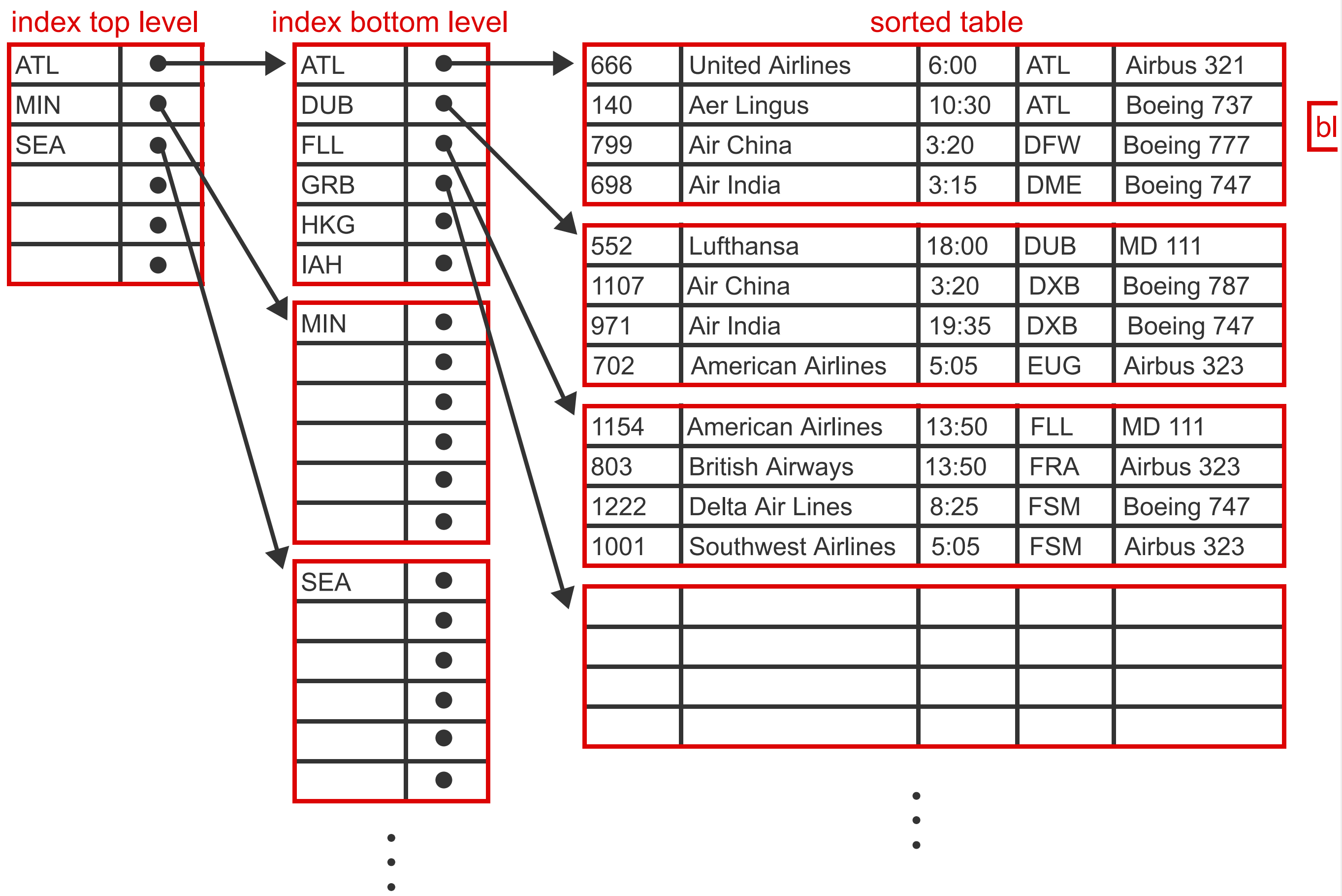
Multi-level indexes

A ***multi-level index*** stores column values and row pointers in a hierarchy. The bottom level of the hierarchy is a sorted single-level index. The bottom level is sparse for primary indexes, or dense for secondary indexes.

Each level above the bottom is a sparse sorted index to the level below. Since all levels above the bottom are sparse, levels rapidly become smaller. The top level always fits in one block.

To locate a row containing an indexed value, the database first reads the top-level block. The database compares the indexed value to entries in the block and locates the next level block containing the value. Continuing in this manner, the database eventually locates the bottom-level block containing the value. The bottom-level block contains a pointer to the correct table block.

PARTICIPATION ACTIVITY 15.1.1: Multi-level indexes.



index top level

ATL	●
MIN	●
SEA	●
	●
	●
	●

index bottom level

ATL	●
DUB	●
FLL	●
GRB	●
HKG	●
IAH	●

MIN	●
	●
	●
	●
	●
	●

SEA	●
	●
	●
	●
	●
	●

sorted table

666	United Airlines	6:00	ATL	Airbus 321
140	Aer Lingus	10:30	ATL	Boeing 737
799	Air China	3:20	DFW	Boeing 777
698	Air India	3:15	DME	Boeing 747

552	Lufthansa	18:00	DUB	MD 111
1107	Air China	3:20	DXB	Boeing 787
971	Air India	19:35	DXB	Boeing 747
702	American Airlines	5:05	EUG	Airbus 323

1154	American Airlines	13:50	FLL	MD 111
803	British Airways	13:50	FRA	Airbus 323
1222	Delta Air Lines	8:25	FSM	Boeing 747
1001	Southwest Airlines	5:05	FSM	Airbus 323

Animation content:

Static figure:
A sorted table has four blocks. Each block contains four rows. Each row contains flight number, airline name, departure time, airport code, and aircraft type. The rows are sorted on airport code.

An index has entries containing an airport code and a pointer. The index has two levels. Entries in both levels are sorted by airport code. Bottom-level entries point to the table block containing the entry's airport code. Top-level entries point to the bottom-level block containing the entry's airport code.

Both index levels are sparse - entries point to index or table blocks rather than index entries or table rows.

Step 1: Flight table is sorted on non-unique column DepartureAirport. The sorted table appears. Airport codes are highlighted.

Step 2: The bottom level is a sparse index on DepartureAirport. If the table is not sorted on the index column, the bottom level must be dense. The bottom index level appears. Each entry is highlighted, along with the first row of the table block that the entry points to.

Step 3: The next higher level is a sparse index to the lower level. The top index level appears. Each entry is highlighted, along with the first entry of the bottom-level block that the entry points to.

Animation captions:

1. Flight table is sorted on non-unique column DepartureAirport.
2. The bottom level is a sparse index on DepartureAirport. If the table is not sorted on the index column, the bottom level must be dense.
3. The next higher level is a sparse index to the lower level.

PARTICIPATION ACTIVITY 15.1.2: Multi-level indexes.

1) The bottom level of a multi-level index is always sparse.

○ True

○ False

2) Levels above the bottom of a multi-level index are always sparse.

○ True

○ False

3) Each column value appears at most once in a multi-level index.

○ True

○ False

4) In a sparse multi-level index, each table block pointer appears exactly once.

○ True

○ False

Number of levels

A dense index has more bottom-level entries than a sparse index, and may have more levels. Assuming a table with 10 million rows and 400 index entries per block, a dense index has three levels:

- Level 3 is dense and has 25,000 blocks = 10 million rows / 400 index entries per block
- Level 2 is sparse and has 63 blocks = 25,000 level 3 blocks / 400 index entries per block
- Level 1 is sparse and has one block containing 63 index entries.

The number of index entries per block is called the ***fan-out*** of a multi-level index. The number of levels in a multi-level index can be computed from fan-out, number of rows, and rows per block:

- For a dense index, number of levels = $\log_{\text{fan-out}}$ (number of rows)
- For a sparse index, number of levels = $\log_{\text{fan-out}}$ (number of rows / rows per block)

In both cases, log is a fractional number and must be rounded up to the nearest integer. Both formulas assume minimal free space in the index.

Dense indexes usually have four levels or less. Sparse indexes usually have three levels or less.

Table 15.1.1: Number of levels.

	rows

dense index		1 million	1 billion
index entries per block	100	3	5
	400	3	4

sparse index		rows	
		1 million	1 billion
index entries per block	100	3	4
	400	2	3

PARTICIPATION ACTIVITY 15.1.3: Number of levels.

Assume a table has 1,000,000 rows. Index entries are 26 bytes, and index blocks are 8 kilobytes. Use a calculator to compute logs.

1) What is the fan-out for a multi-level index?

○ Approximately 200

○ Approximately 300

○ Approximately 400

2) How many levels does a dense multi-level index have?

○ 2

○ 3

○ 4

3) If table blocks are 2 kilobytes and rows are 100 bytes, how many levels does a sparse multi-level index have?

○ 2

○ 3

○ 4

Query processing

Multi-level indexes are faster than single-level indexes on most queries. Consider the following scenario:

- A table has 10 million rows.
- Each row is 100 bytes.
- Each index entry is 10 bytes.
- Table and index blocks are 4 kilobytes.

The table contains 250,000 blocks = 10 million rows × 100 bytes per row / 4,000 bytes per block.

A dense, single-level index contains 25,000 blocks = 10 million entries × 10 bytes per entry / 4,000 bytes per block.

A dense, multi-level index contains 3 levels = $\log_{400 \text{ entries per index block}}$ 10,000,000 rows, rounded up.

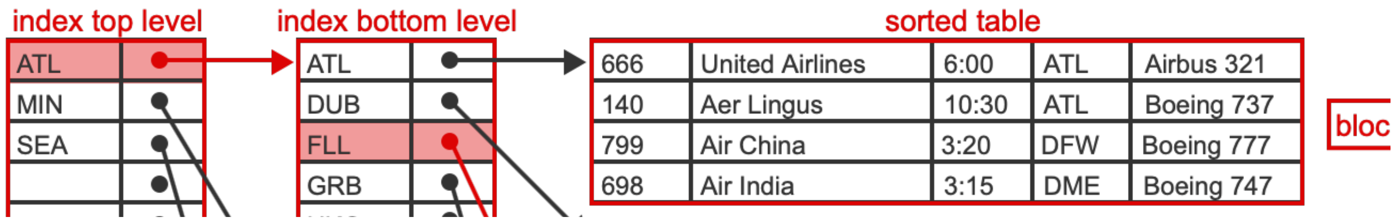
A query searches for rows containing a specific value of an indexed column. Assuming query hit ratio is low:

- A table scan reads at most 250,000 table blocks.
- A single-level index scan reads at most 25,000 index blocks plus a few table blocks.
- A binary search of a sorted, single-level index reads at most 15 index blocks (= $\log_2$ 25,000, rounded up) plus a few table blocks.
- A multi-level index search reads 3 index blocks plus a few table blocks.

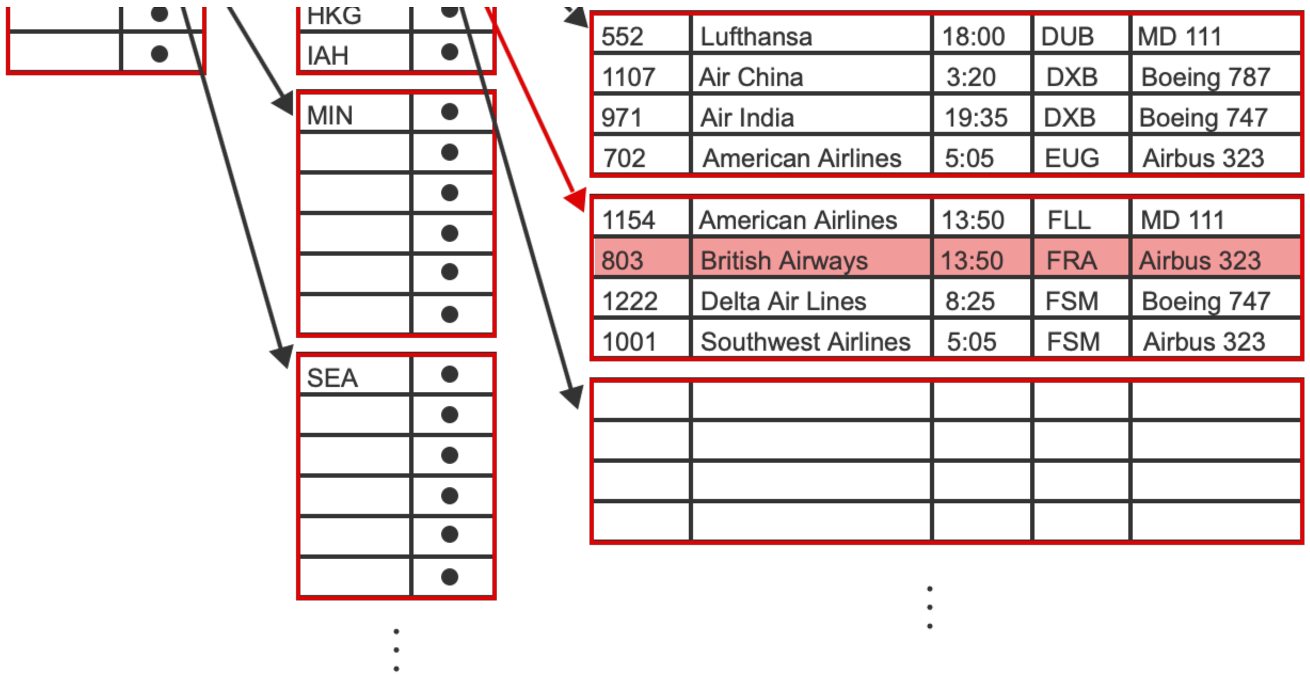
The multi-level index search reads one index block per level. Usually the top two levels are small and retained in memory. Since the index has three levels, the query reads just one index block from storage media.

Because multi-level indexes are faster than single-level indexes on most queries, databases commonly use multi-level rather than single-level indexes.

PARTICIPATION ACTIVITY 15.1.4: Query processing with a multi-level index.



666	United Airlines	6:00	ATL	Airbus 321
140	Aer Lingus	10:30	ATL	Boeing 737
799	Air China	3:20	DFW	Boeing 777
698	Air India	3:15	DME	Boeing 747



552	Lufthansa	18:00	DUB	MD 111
1107	Air China	3:20	DXB	Boeing 787
971	Air India	19:35	DXB	Boeing 747
702	American Airlines	5:05	EUG	Airbus 323

1154	American Airlines	13:50	FLL	MD 111
803	British Airways	13:50	FRA	Airbus 323
1222	Delta Air Lines	8:25	FSM	Boeing 747
1001	Southwest Airlines	5:05	FSM	Airbus 323

```
SELECT AirlineName
FROM Flight
WHERE DepartureAirport = 'FRA';
```

Animation content:

Static figure:
A sorted table has four blocks. Each block contains four rows. Each row contains flight number, airline name, departure time, airport code, and aircraft type. The rows are sorted on airport code.

An index has entries containing an airport code and a pointer. The index has two levels. Entries in both levels are sorted by airport code. Bottom-level entries point to the table block containing the entry's airport code. Top-level entries point to the bottom-level block containing the entry's airport code.

Both index levels are sparse - entries point to index or table blocks rather than index entries or table rows.

An SQL statement appears.
Begin SQL code:
SELECT AirlineName
FROM Flight
WHERE DepartureAirport = 'FRA';
End SQL code.

Step 1: A query searches for rows containing 'FRA'. The WHERE clause is highlighted.

Step 2: The database reads the index's top level. 'FRA' is between 'ATL' and 'MIN'. FRA appears between top-level entries for ATL and MIN. The top-level ATL entry is highlighted.

Step 3: The database follows the 'ATL' pointer and reads the bottom-level block. 'FRA' is between 'FLL' and 'GRB'. The top-level entry for ATL points to a bottom-level block containing entries for FLL and GRM. FRA appears between the FLL and GRM entries. The bottom-level FLL entry is highlighted.

Step 4: The database follows the 'FLL' pointer, reads the table block, and locates the row containing 'FRA'. FRA appears next to the table block that the bottom-level index block containing FLL points to. The row in this table block containing FRA is highlighted.

Animation captions:

1. A query searches for rows containing 'FRA'.
2. The database reads the index's top level. 'FRA' is between 'ATL' and 'MIN'.
3. The database follows the 'ATL' pointer and reads the bottom-level block. 'FRA' is between 'FLL' and 'GRB'.
4. The database follows the 'FLL' pointer, reads the table block, and locates the row containing 'FRA'.

PARTICIPATION ACTIVITY 15.1.5: Query processing.

Match the search type to the maximum number of blocks read.

If unable to drag and drop, refresh the page.

Single-level index binary search **Single-level index scan** **Table scan**

Multi-level sparse index search **Multi-level dense index search**

	Number of rows / rows per table block
	Number of index blocks plus

	referenced table blocks.
	log base 2 (number of index blocks) plus referenced table blocks.
	log base fan-out (number of rows), rounded up, plus referenced table blocks.
	log base fan-out (number of rows / rows per table block), rounded up, plus referenced table blocks.

Reset

Balanced indexes

Each path from the top-level block to a bottom-level block is called a ***branch***. Multi-level indexes are called ***balanced*** when all branches are the same length and ***imbalanced*** when branches are different lengths.

Imbalanced indexes are undesirable, since processing time is unpredictable. If a query follows a long branch, the query is relatively slow. For this reason, inserts are managed to maintain balanced indexes:

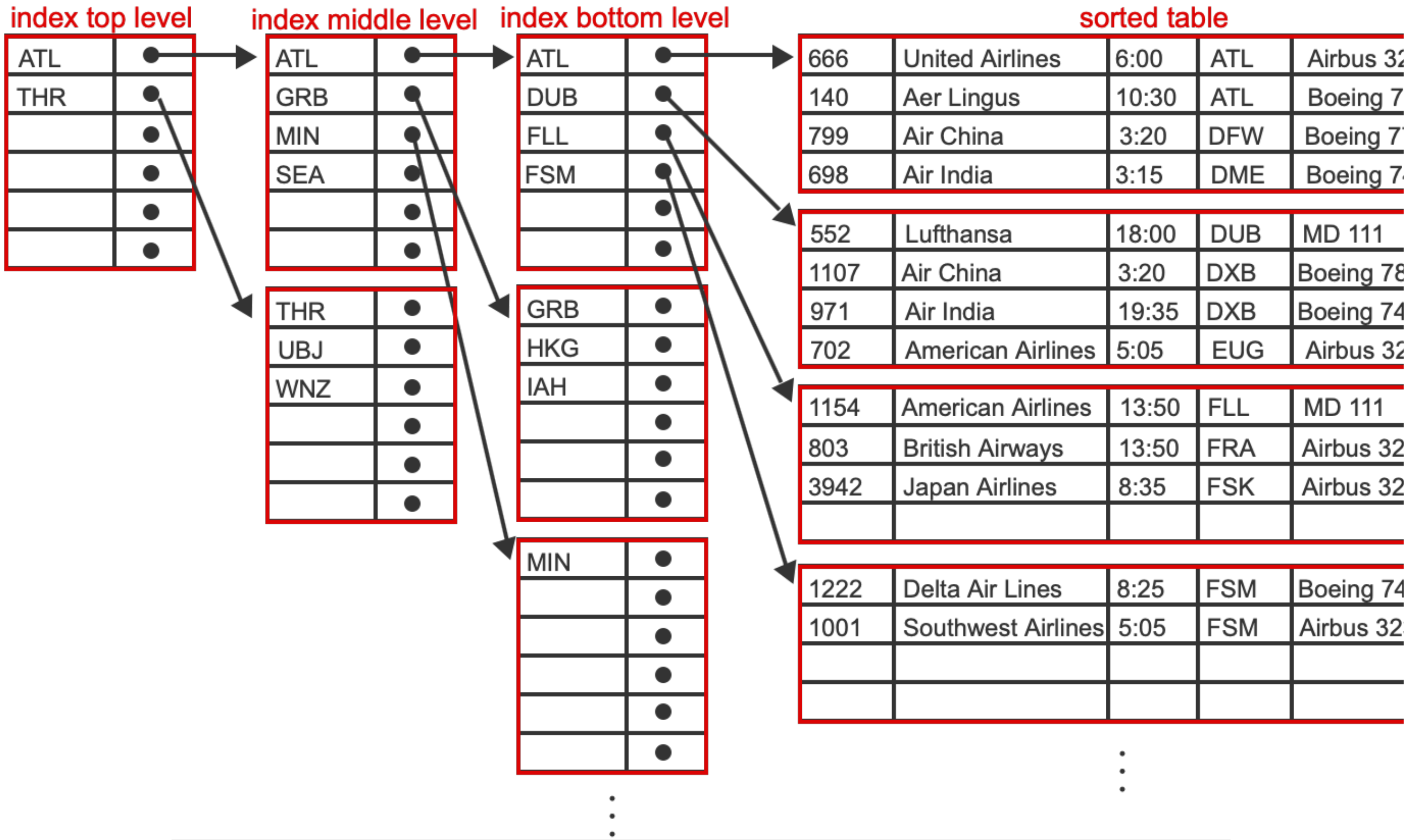
1. In a dense index, inserts always generate new bottom-level index entries. In a sparse index, inserts generate new bottom-level index entries when table blocks split.
2. If the new index entry goes in a full index block, the block splits. Half of the rows move to the new block, creating space for the entry.
3. The new block in the bottom level generates a new index entry the next level up. If the block in the next level up is full, the block splits and the process repeats.
4. If blocks are full at all index levels, the split propagates to the top level. In this case, the top-level block splits and a new level is created.

New levels are always added at the top of the hierarchy rather than the bottom of one branch. As a result, all branches are always the same length, and the index is always balanced.

Deletes may cause block mergers. Block mergers are the reverse of block splits and potentially eliminate the top level of the index. Consequently, deletes also maintain a balanced index.

Updates to an indexed column behave like a delete of the initial value followed by an insert of a new value. Since updates are implemented as deletes and inserts, updates also leave the index balanced.

PARTICIPATION ACTIVITY | 15.1.6: Balancing a sparse index when inserting.



ATL	●
THR	●
	●
	●
	●
	●

ATL	●
GRB	●
MIN	●
SEA	●
	●
	●

THR	●
UBJ	●
WNZ	●
	●
	●
	●

ATL	●
DUB	●
FLL	●
FSM	●
	●
	●

GRB	●
HKG	●
IAH	●
	●
	●
	●

MIN	●
	●
	●
	●
	●
	●

666	United Airlines	6:00	ATL	Airbus 32
140	Aer Lingus	10:30	ATL	Boeing 7
799	Air China	3:20	DFW	Boeing 7
698	Air India	3:15	DME	Boeing 7

552	Lufthansa	18:00	DUB	MD 111
1107	Air China	3:20	DXB	Boeing 78
971	Air India	19:35	DXB	Boeing 74
702	American Airlines	5:05	EUG	Airbus 32

1154	American Airlines	13:50	FLL	MD 111
803	British Airways	13:50	FRA	Airbus 32
3942	Japan Airlines	8:35	FSK	Airbus 32

1222	Delta Air Lines	8:25	FSM	Boeing 74
1001	Southwest Airlines	5:05	FSM	Airbus 32

```
INSERT INTO Flight
VALUES (3942, 'Japan Airlines', '8:35', 'FSK', 'Airbus 321')
```

Animation content:

Static figure:
A sorted table has four blocks. Each block contains four rows. Each row contains flight number, airline name, departure time, airport code, and aircraft type. The rows are sorted on airport code.

An index has entries containing an airport code and a pointer. The index has three levels. Entries in all levels are sorted by airport code. Bottom-level entries point to the table block containing the entry's airport code. Middle-level entries point to the bottom-level block containing the entry's airport code. Top-level entries point to the middle-level block containing the entry's airport code.

All index levels are sparse - entries point to index or table blocks rather than index entries or table rows.

An SQL statement appears.
Begin SQL code:
INSERT INTO Flight
VALUES (3942, 'Japan Airlines', 8:35, 'FSK', 'Airbus 321');
End SQL code.

Step 1: The Flight table is sorted on DepartureAirport. A row is inserted with DepartureAirport 'FSK'. The index, table, and statement appear. The index has two levels, labeled top level and bottom level.

Step 2: The table block is full and splits to create space for the new row. Table block 2 splits into two blocks. The last two rows of block 2 move to the new block 3. A new row is inserted to the new space created in table block 2:
3942, Japan Airlines, 8:35, FSK, Airbus 321

Step 3: The new table block generates a new entry in the index's bottom level. The airport code FSM, from the first row of the new block 3, appears between the FLL and GRB entries in the bottom index level.

Step 4: The bottom-level index block is full and splits. Bottom-level index block 0 splits into two blocks. The last three entries of index block 0 move to the new index block 1. A new entry for FSM is inserted to the new space created in index block 0. The new entry points to the new table block 2, containing the new row.

Step 5: The new bottom-level block generates a new entry for the index's top level. Airport code GRB, from the first entry of the new index block 1, appears between entries for ATL and MIN of the index top level.

Step 6: The top-level index is full and splits. The index top level is renamed index middle level. A new level appears with caption index top level. Middle-level block 0 splits into two blocks. The last three entries of middle-level block 0 move to the new middle-level block 1. A new GRB entry is inserted to middle-level block 0, pointing to the corresponding bottom-level block. The new top-level index has one block with entries for ATL and THR, pointing to the corresponding middle-level blocks.

Animation captions:

1. The Flight table is sorted on DepartureAirport. A row is inserted with DepartureAirport 'FSK'.

2. The table block is full and splits to create space for the new row.
3. The new table block generates a new entry in the index's bottom level.
4. The bottom-level index block is full and splits.
5. The new bottom-level block generates a new entry for the index's top level.
6. The top-level index is full and splits. A new top level is created containing entries for two blocks in the middle level.

PARTICIPATION ACTIVITY | 15.1.7: Balanced indexes.

A table has a multi-level index with no free space in any index blocks.

1) If the index is sparse, an insert to the table always generates a new index level.

○ True

○ False

2) If the index is dense, an insert to the table always generates a new index level.

○ True

○ False

3) If the index is dense, an update to the indexed column always generates a new index level.

○ True

○ False

B-tree and B+tree indexes

The balanced multi-level index described above is called a B+tree. B+tree structure is derived from an earlier approach called a B-tree. The two differ as follows:

- ***B+tree***. All indexed values appear in the bottom level. Pointers to table blocks appear only in the bottom level. Since some indexed values also appear in higher levels, values are occasionally repeated in the index.
- ***B-tree***. If an indexed value appears in a higher level, the value is not repeated at lower levels

Instead, a pointer to the corresponding table block appears in the higher level along with the value.

B-trees are more compact than B+trees since index values are not repeated. However, B+trees are simpler, since all pointers to table blocks appear in the same (bottom) level. The B+tree structure has two benefits:

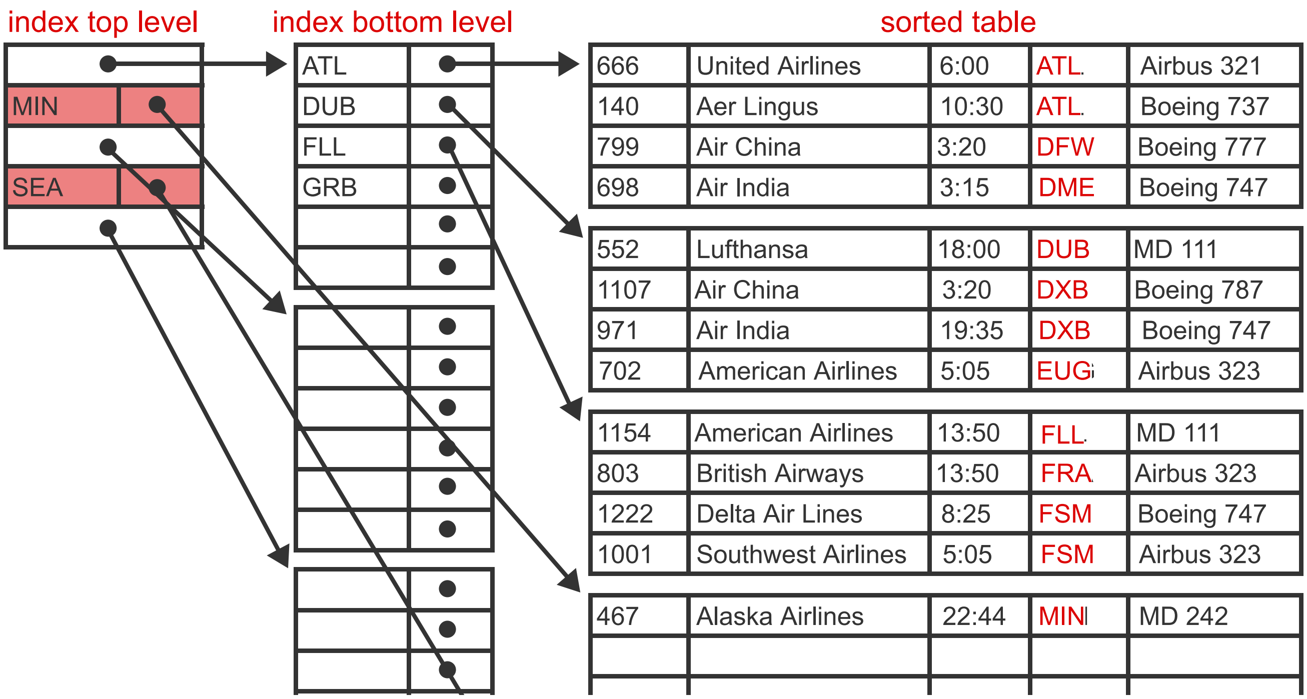
- The bottom level of a B+tree is a single-level index and can be scanned or searched.
- In a B-tree, inserts, updates, and deletes may cause a table pointer to change levels, which is hard to implement. B+trees do not have this problem, since table pointers are always in the bottom level.

Because of the advantages above, multi-level indexes are usually implemented as B+trees.

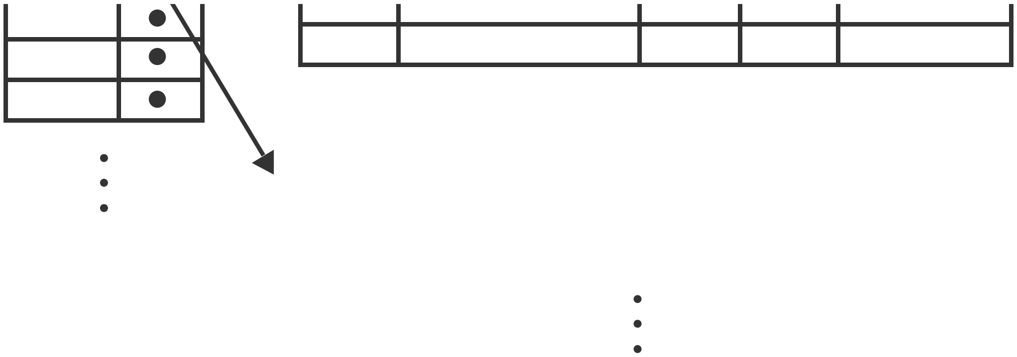
Terminology

Although most multi-level indexes are implemented as B+trees, the term ***B-tree*** *is commonly used and often refers to a B+tree structure.* ***B+tree*** *is commonly written as B+-tree or B$^+$-tree.*

PARTICIPATION ACTIVITY 15.1.8: B-tree index.



666	United Airlines	6:00	ATL	Airbus 321
140	Aer Lingus	10:30	ATL	Boeing 737
799	Air China	3:20	DFW	Boeing 777
698	Air India	3:15	DME	Boeing 747
552	Lufthansa	18:00	DUB	MD 111
1107	Air China	3:20	DXB	Boeing 787
971	Air India	19:35	DXB	Boeing 747
702	American Airlines	5:05	EUG	Airbus 323
1154	American Airlines	13:50	FLL	MD 111
803	British Airways	13:50	FRA	Airbus 323
1222	Delta Air Lines	8:25	FSM	Boeing 747
1001	Southwest Airlines	5:05	FSM	Airbus 323
467	Alaska Airlines	22:44	MIN	MD 242



Animation content:

Static figure:
A sorted table has four blocks. Each block contains four rows. Each row contains flight number, airline name, departure time, airport code, and aircraft type. The rows are sorted on airport code.

An index has entries containing an airport code and a pointer. The index has two levels. Entries in both levels are sorted by airport code. Bottom-level entries point to the table block containing the entry's airport code. The bottom level is identical to the bottom level in prior animations in this section.

The top level is different from prior animations in this section. The top level has five entries. Entries 0, 2, and 4 contain a pointer only, pointing to bottom-level index blocks 0, 1, and 2, respectively. Entries 1 and 3 contain an airport code and a pointer. These entries point to the table block that begins with the corresponding airport code.

Both index levels are sparse - entries point to index or table blocks rather than index entries or table rows.

Step 1: The bottom level of a B-tree is like the bottom level of a B+tree. The table and index bottom level appear.

Step 2: Higher levels of a B-tree alternate index block pointers and table block pointers. The index top level appears. Each top-level entry is highlighted, along with the table or index block that the entry points to.

Step 3: Entries that appear in higher levels are not repeated in the bottom level. Top-level entries 1 and 3, containing an airport code and a pointer to a table block, are highlighted. These entries point directly to the corresponding table block, and do not appear in the index bottom level.

Animation captions:

1. The bottom level of a B-tree is like the bottom level of a B+tree.
2. Higher levels of a B-tree alternate index block pointers and table block pointers.
3. Entries that appear in higher levels are not repeated in the bottom level.

PARTICIPATION ACTIVITY | 15.1.9: B-tree and B+tree indexes.

1) Which type of index can have table block pointers at any level?

○ B-tree

○ B+tree

○ Both B-tree and B+tree

2) In informal use, the term 'B-tree' commonly means:

○ B-tree index

○ B+tree index

○ Either a B-tree or a B+tree index

3) Which index structure enables an index scan using the bottom level only?

○ B-tree

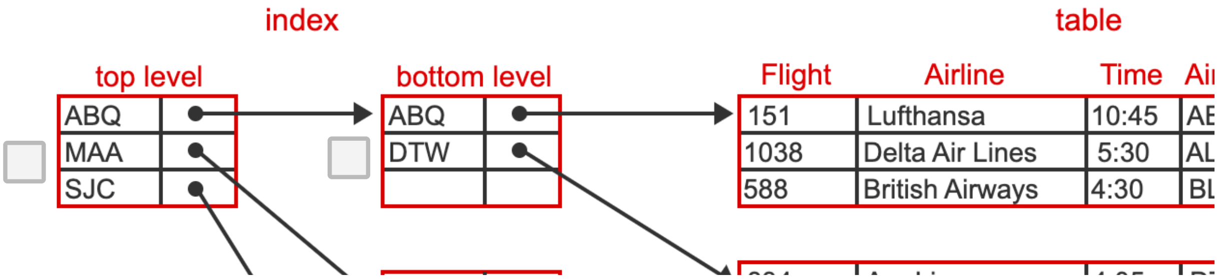
○ B+tree

○ Both B-tree and B+tree

CHALLENGE ACTIVITY | 15.1.1: Multi-level indexes.

544874.3500394.qx3zqy7

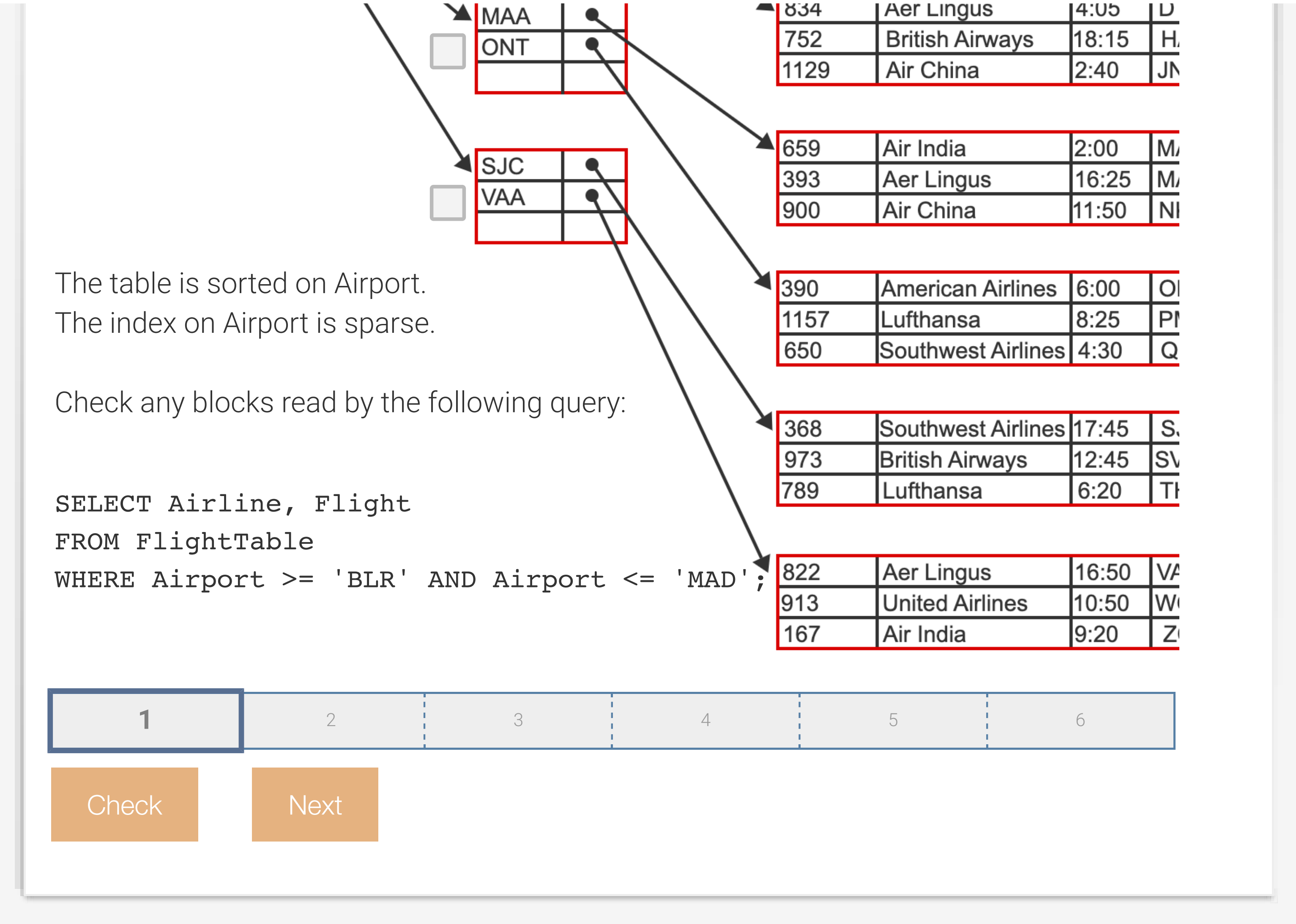
Start



ABQ	●
MAA	●
SJC	●

ABQ	●
DTW	●

Flight	Airline	Time	Ai
151	Lufthansa	10:45	AE
1038	Delta Air Lines	5:30	AL
588	British Airways	4:30	BL



MAA	●
ONT	●

SJC	●
VAA	●

834	Aer Lingus	4:05	D
752	British Airways	18:15	H
1129	Air China	2:40	JN

659	Air India	2:00	M
393	Aer Lingus	16:25	M
900	Air China	11:50	NI

390	American Airlines	6:00	O
1157	Lufthansa	8:25	PI
650	Southwest Airlines	4:30	Q

368	Southwest Airlines	17:45	S
973	British Airways	12:45	SV
789	Lufthansa	6:20	TI

822	Aer Lingus	16:50	VA
913	United Airlines	10:50	W
167	Air India	9:20	Z

The table is sorted on Airport.
The index on Airport is sparse.

Check any blocks read by the following query:

```
SELECT Airline, Flight
FROM FlightTable
WHERE Airport >= 'BLR' AND Airport <= 'MAD';
```