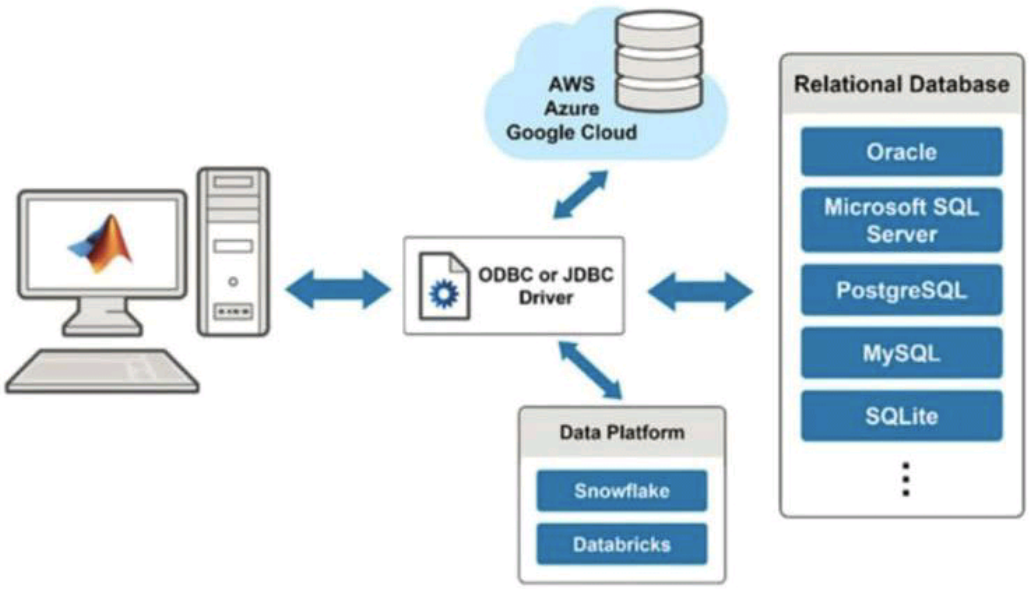
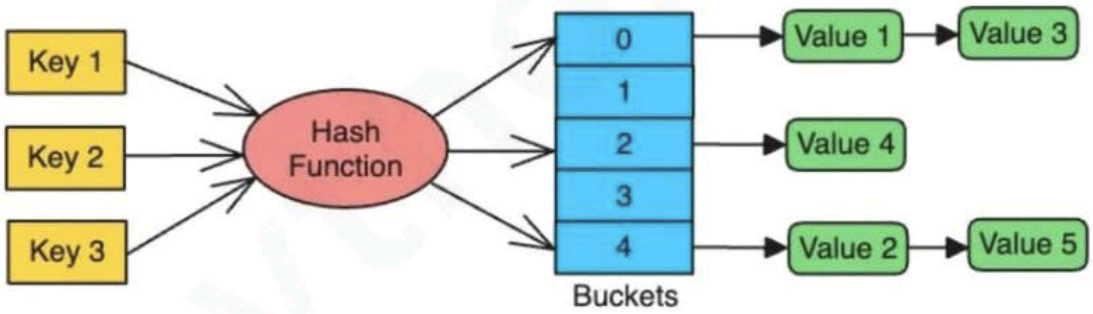
7 Data Structures That Make Databases Fast and Scalable



1. Hash Indexes

A hash index is a data structure that maps keys to values using a hash function.

The hash function converts a key into an integer, which is used as an index in a hash table (buckets) to locate the corresponding value.



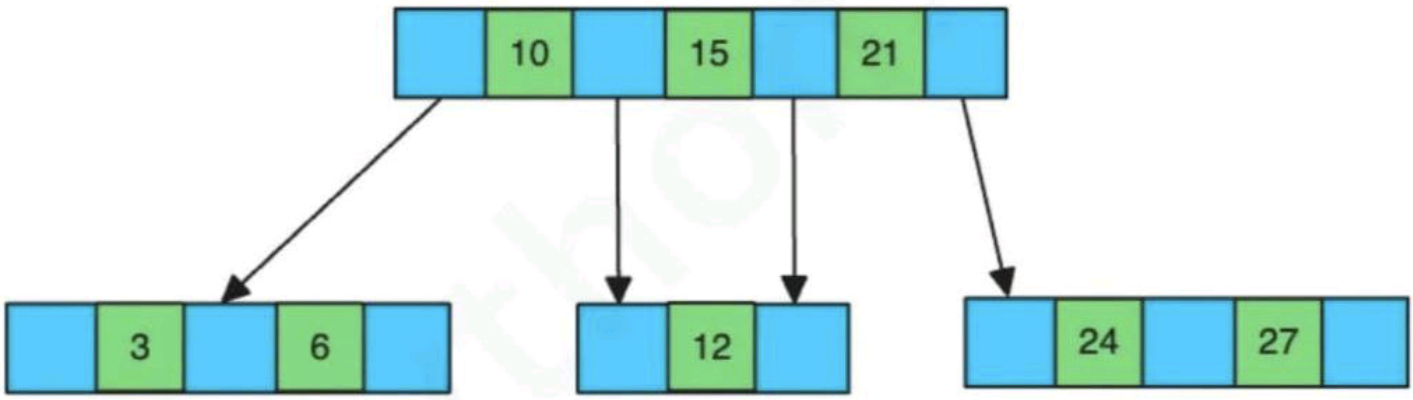
This structure provides O(1) average-time complexity for insertions, deletions, and lookups.

Hash indexes are widely used in key-value stores (eg., DynamoDB), and caching systems (eg., Redis).

2. B-Trees

A B-tree is a self-balancing tree that optimizes reads, writes, and queries on large datasets by storing multiple keys per node and minimizing disk I/O.

It auto-balances during insertions and deletions.



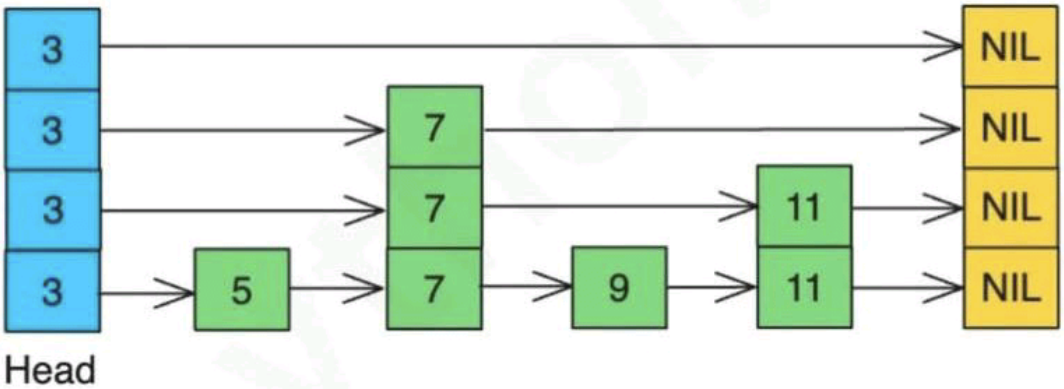
B-Trees are widely used for indexing in relational databases (eg., MySQL).

While many NoSQL databases favor LSM Trees for write-heavy workloads, some use B-Trees for read-heavy scenarios or as part of their indexing strategy.

3. Skip Lists

A skip list is a probabilistic data structure that enhances linked lists with shortcut levels, enabling fast search, insertion, and deletion.

It offers performance similar to balanced trees but is simpler to implement and manage.



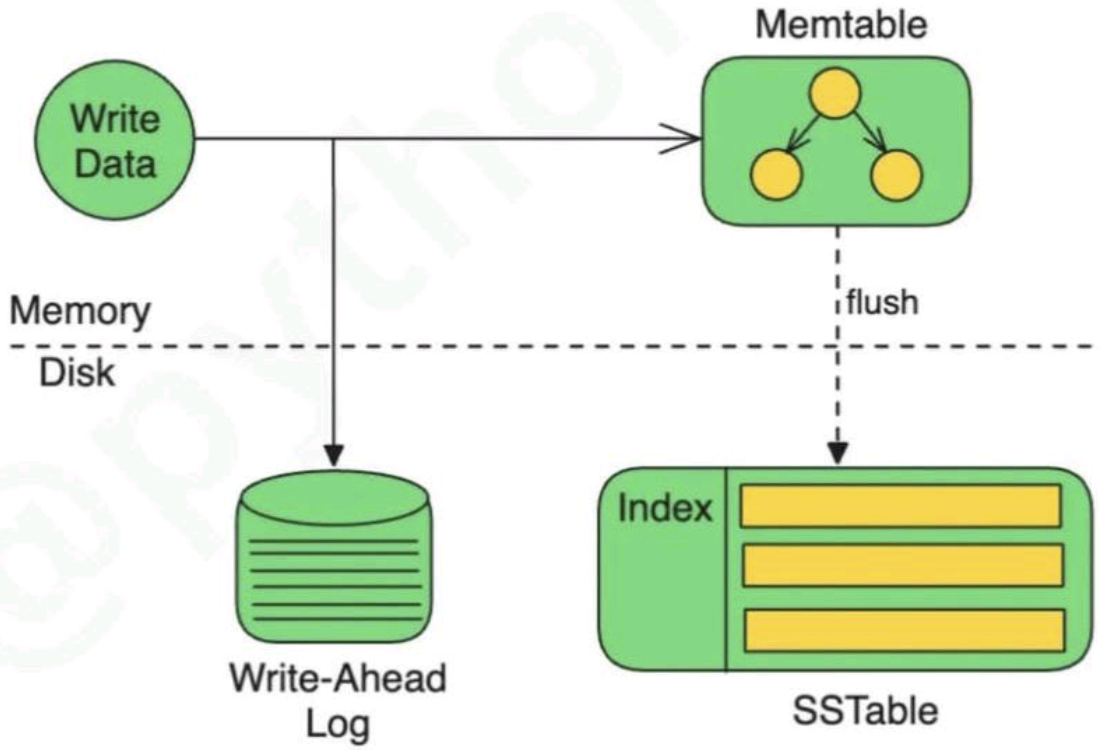
Skip lists are ideal for in-memory storage and dynamic datasets with frequent updates.

Redis uses skip lists for sorted sets (ZSET), enabling fast operations and maintaining sorted order.

4. SSTables

An SSTable is an immutable on-disk structure storing sorted key-value pairs, used in databases like Cassandra, RocksDB, and HBase.

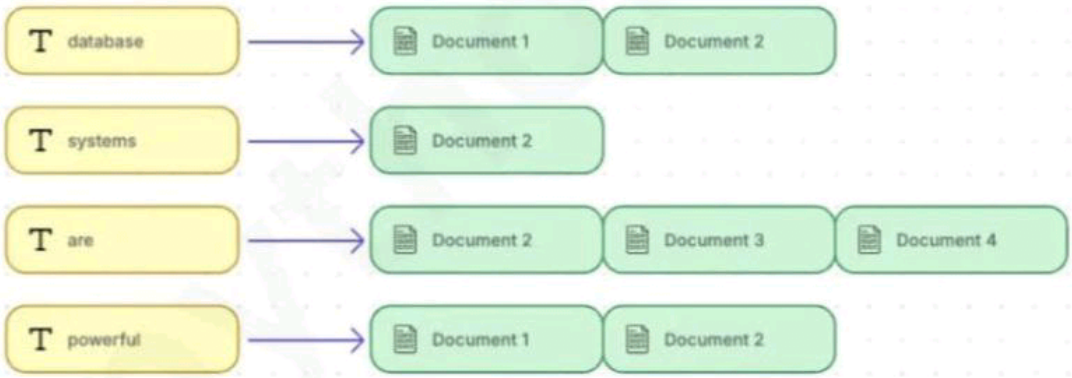
SSTables in LSM Trees optimize writes, lookups, and range queries.



5. Inverted Index

An inverted index maps terms to the documents or locations where they appear.

An inverted index maps terms to the documents they appear in, reversing the conventional index relationship.

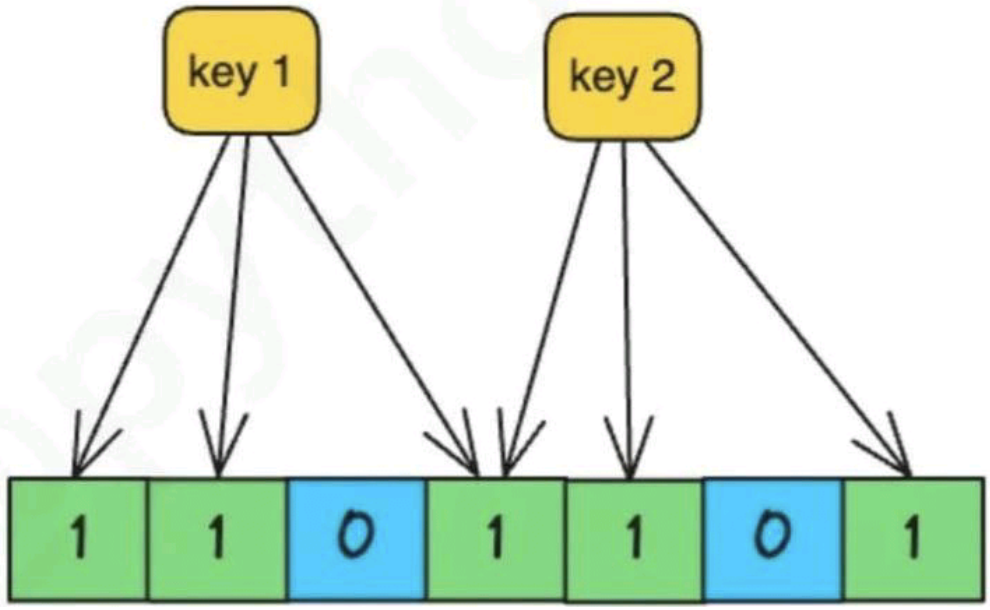


Inverted indexes are widely used in databases, search engines, and information retrieval systems to enable efficient keyword lookups, Boolean queries, and relevance ranking.

6. Bloom Filters

A Bloom Filter is a space-efficient, probabilistic data structure that answers the question: "Does this element exist in a set?"

It starts as a bit array of size mm, initialized with all bits set to 0, and uses kk independent hash functions to map elements to positions in the bit array.



7. Bitmap Indexes

A bitmap index encodes column values as bitmaps, where each bitmap represents a unique value.

Each bit in the bitmap represents whether a row in the dataset contains that value.

Consider a dataset with a column Color:

Dataset

Row	Color
1	Red
2	Blue
3	Green
4	Red
5	Green

Bitmap

Value	Bitmap
Red	1 0 0 1 0
Blue	0 1 0 0 0
Green	0 0 1 0 1