从数据结构角度看索引

To D 研发 不猛

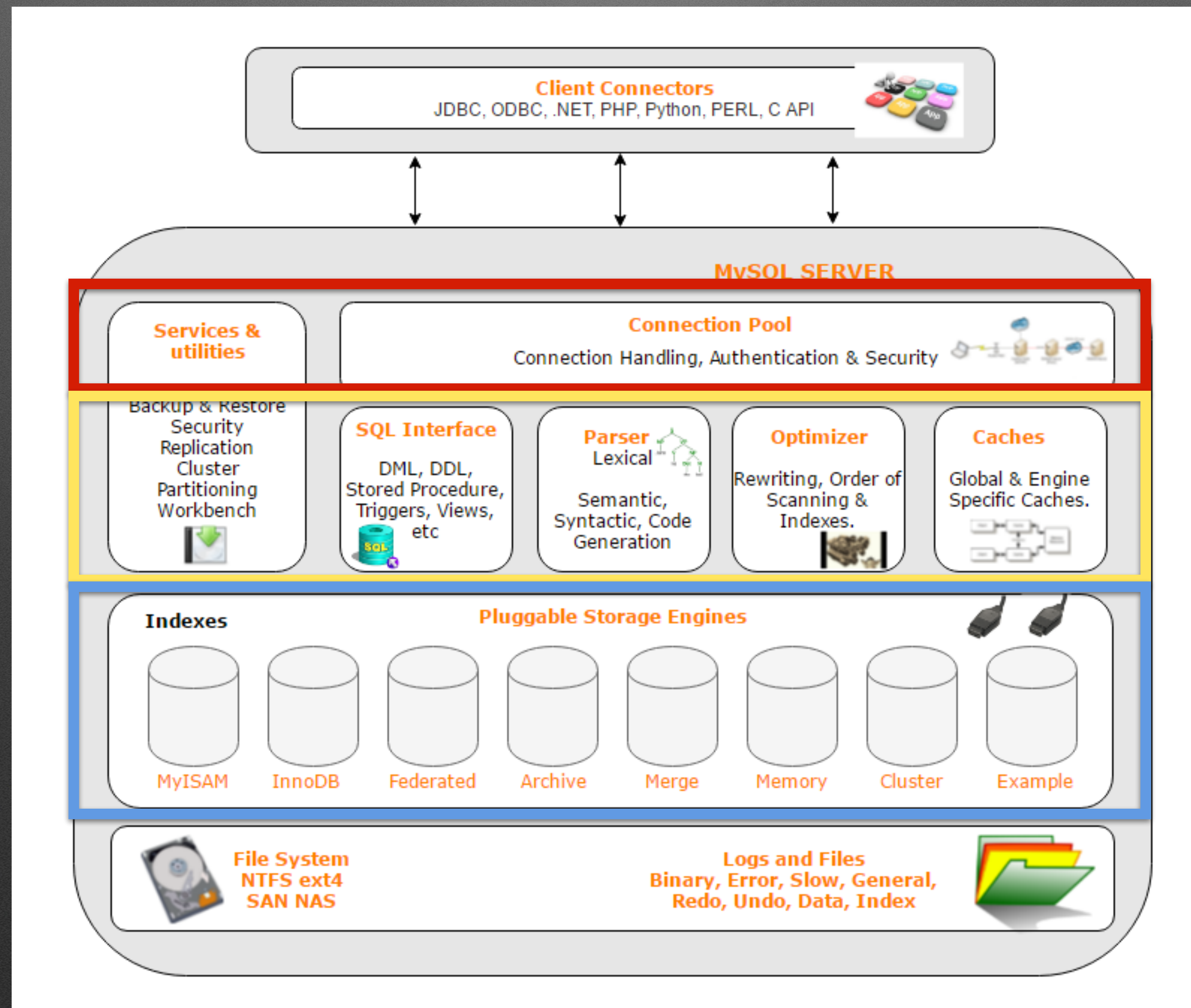
目录

1. 知识储备
2. 索引是什么，有什么用
3. 索引的各种类型
    i. 索引的数据结构
    ii. 索引的应用场景
    iii. 索引的案例分析

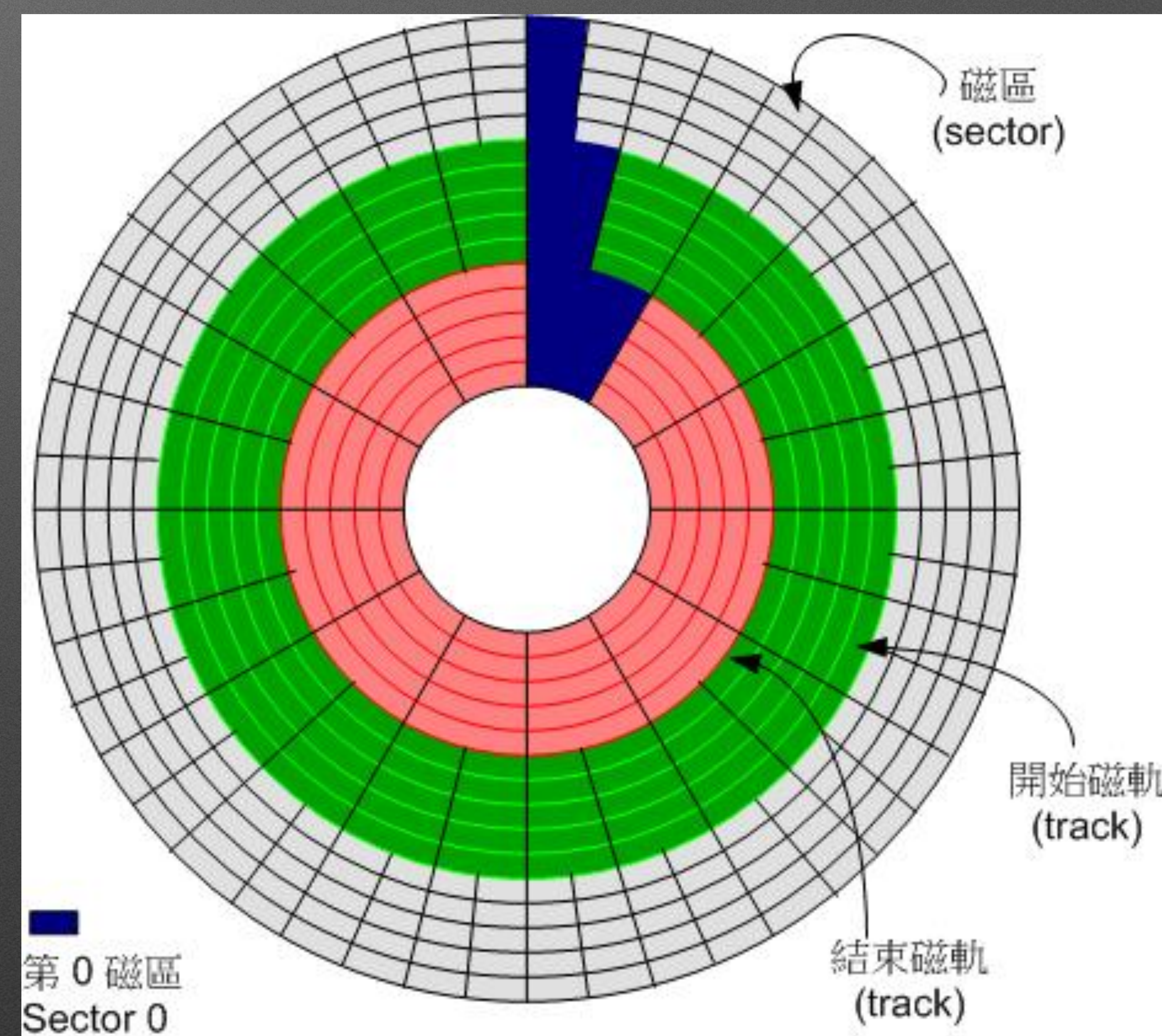
知识储备

- MySQL Server 架构
- 磁盘存取原理
- 局部性原理

MySQL Server 架构

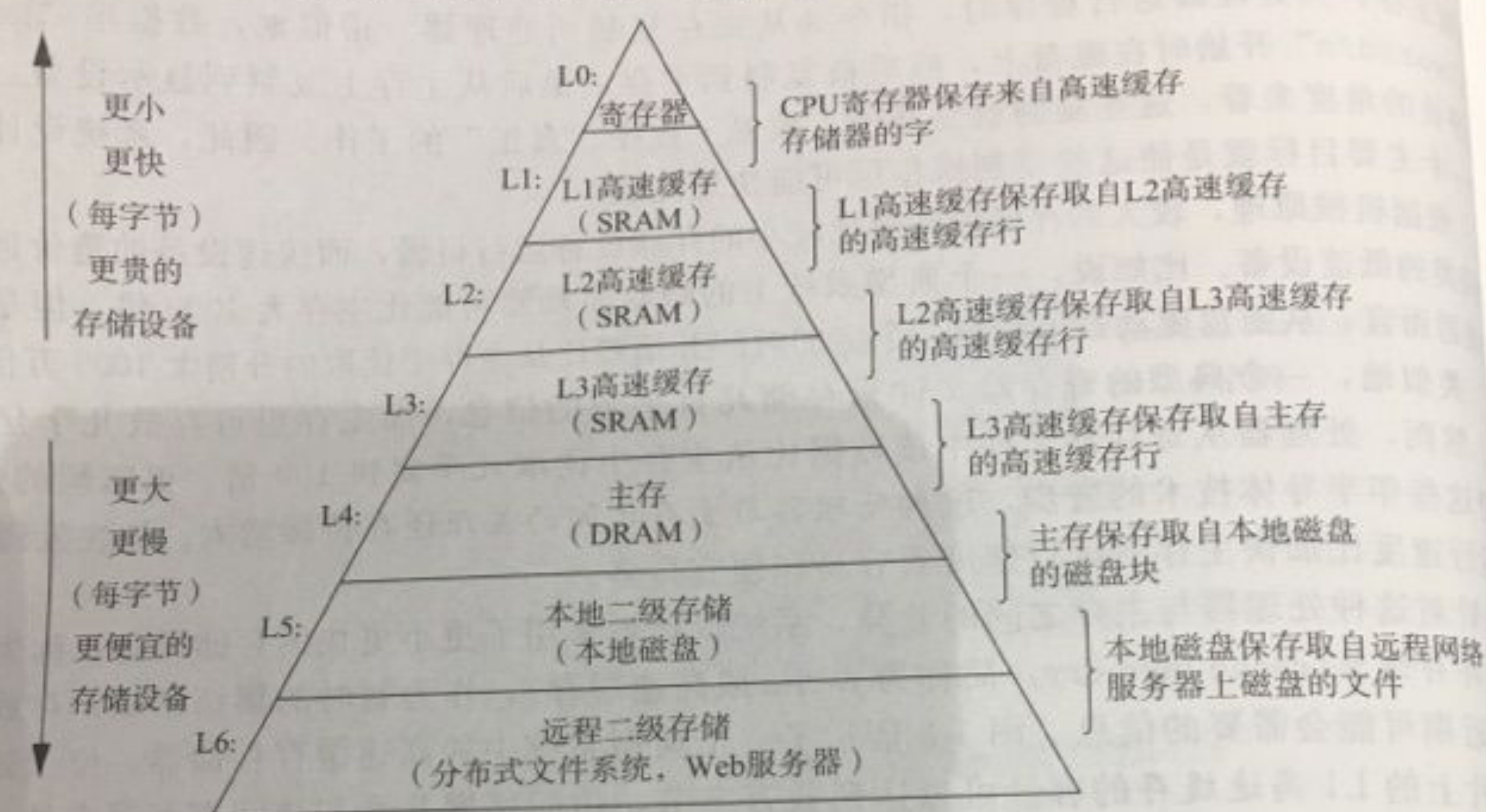


磁盤存取原理



局部性原理

- 时间局部性：如果一个信息项正在被访问，那么在近期它很可能还会被再次访问。
- 空间局部性：在最近的将来将用到的信息很可能与现在正在使用的信息在空间地址上是临近的。



组织成了一个存储器层次结构，如图 1-9 所示。在这个层次结构中，从上至下，设备的访问速度越来越慢、容量越来越大，并且每字节的造价也越来越便宜。寄存器文件在层次结构中位于最顶部，也就是第 0 级或记为 L0。这里我们展示的是三层高速缓存 L1 到 L3，占据存储器层次结构的第 1 层到第 3 层。主存在第 4 层，以此类推。

图 1-9　一个存储器层次结构的示例

存储器层次结构的主要思想是上一层的存储器作为低一层存储器的高速缓存。因此，寄存器文件就是 L1 的高速缓存，L1 是 L2 的高速缓存，L2 是 L3 的高速缓存，L3 是主存的高速缓存，而主存又是磁盘的高速缓存。在某些具有分布式文件系统的网络系统中，本地磁盘就是存储在其他系统中磁盘上的数据的高速缓存。

正如可以运用不同的高速缓存的知识来提高程序性能一样，程序员同样可以利用对整个存储器层次结构的理解来提高程序性能。第 6 章将更详细地讨论这个问题。

1.7　操作系统管理硬件

让我们回到 hello 程序的例子。当 shell 加载和运行 hello 程序时，以及 hello 程

索引是什么，有什么用

- 索引是存储引擎用于快速找到记录的一种数据结构。
- 索引的优点

1. 大大减少了服务器需要扫描的数据；
2. 可以帮助服务器避免排序和临时表；
3. 可以将随机 I/O 变为顺序 I/O；

索引是什么，有什么用

```
mysql> SELECT COUNT(*) FROM user;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (0.12 sec)

mysql> SELECT * FROM user WHERE name = '5ad35e92a06243028ce280e70bc00bc0';
+--------+----------------------------------+-------------+
| id     | name                             | phone       |
+--------+----------------------------------+-------------+
| 123456 | 5ad35e92a06243028ce280e70bc00bc0 | 58293043800 |
+--------+----------------------------------+-------------+
1 row in set (0.28 sec)

mysql> CREATE INDEX idx_name ON user(`name`);
Query OK, 0 rows affected (2.70 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM user WHERE name = '5ad35e92a06243028ce280e70bc00bc0';
+--------+----------------------------------+-------------+
| id     | name                             | phone       |
+--------+----------------------------------+-------------+
| 123456 | 5ad35e92a06243028ce280e70bc00bc0 | 58293043800 |
+--------+----------------------------------+-------------+
1 row in set (0.00 sec)

mysql>
```

索引的各种类型

- InnoDB 的 B-Tree 索引
- Memory 的哈希索引
- MyISAM 的全文索引
- MyISAM 的 R-Tree 索引
- ……

哈希索引的数据结构

- 哈希索引基于哈希表实现，对于每一行数据，存储引擎都会对所有的索引列计算一个哈希码。
- 哈希索引将所有的哈希码存储在索引中，同时在哈希表中保存指向每个数据行的指针。

哈希索引的数据结构

```
SELECT * FROM test_hash
WHERE key1 = ? AND key2 = ?
```

SQL

```
function hash(key1, key2){
    ......
}
```

哈希函数

HASHCODE	POINTER
2323	0xDCF9
3434	0xF349
5656	0xC0A8

哈希表

id	key1	key2
1	4317d···	58hwf···
2	802f9···	689hf···
3	82baa···	7gjdn···

数据行

哈希索引的应用场景

- 哈希索引只支持等值比较查询（=、IN、<=>），不支持任何的范围查询（>、<、BETWEEN）。
- 哈希索引的数据不是按照顺序存储的，所以哈希索引不能被应用于排序操作（ORDER BY）。
- 哈希索引始终使用索引列的全部内容来计算哈希值，所以哈希索引不能被应用于匹配部分索引列的查询。
- 哈希索引只包含哈希值和行指针，所以哈希索引不能被应用于覆盖索引。

哈希索引的应用场景

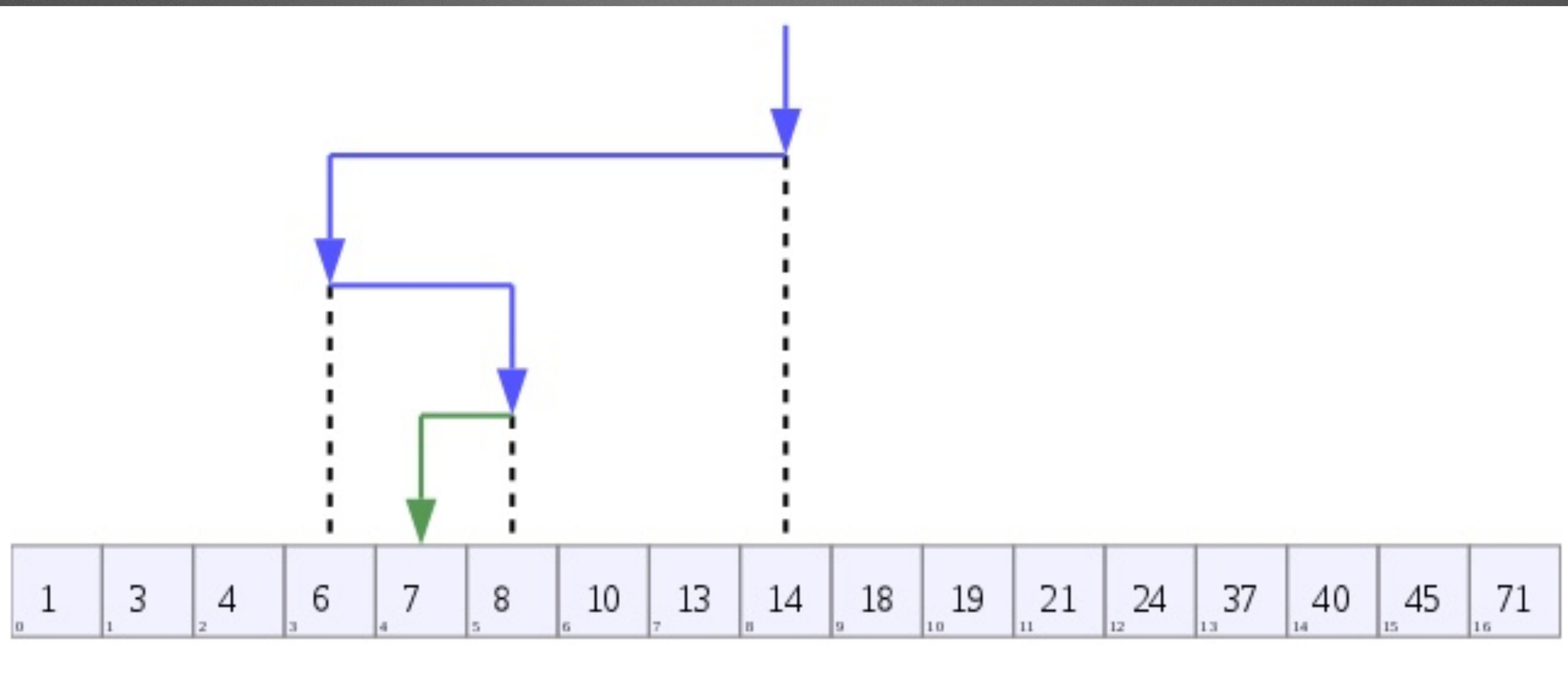
```
mysql> SHOW INDEX FROM test_hash;
+-----------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| Table     | Non_unique | Key_name     | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible |
+-----------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| test_hash |          1 | idx_key1_key2 |            1 | key1        | NULL      |        NULL |     NULL |   NULL |      | HASH       |         |               | YES     |
| test_hash |          1 | idx_key1_key2 |            2 | key2        | NULL      |           0 |     NULL |   NULL |      | HASH       |         |               | YES     |
+-----------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
2 rows in set (0.00 sec)

mysql> EXPLAIN SELECT * FROM test_hash WHERE key1 = '8ca8785a341a11e9995198e6bbf391fd' AND key2 = '8ca878a0341a11e9995198e6bbf391fd';
+----+-------------+-----------+------------+------+---------------+---------------+---------+-------------+------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys | key           | key_len | ref         | rows | filtered | Extra |
+----+-------------+-----------+------------+------+---------------+---------------+---------+-------------+------+----------+-------+
|  1 | SIMPLE      | test_hash | NULL       | ref  | idx_key1_key2 | idx_key1_key2 | 192     | const,const |   10 |   100.00 | NULL  |
+----+-------------+-----------+------------+------+---------------+---------------+---------+-------------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM test_hash WHERE key1 = '8ca8785a341a11e9995198e6bbf391fd';
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | test_hash | NULL       | ALL  | idx_key1_key2 | NULL | NULL    | NULL |   10 |    10.00 | Using where |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql>
```

二分查找算法

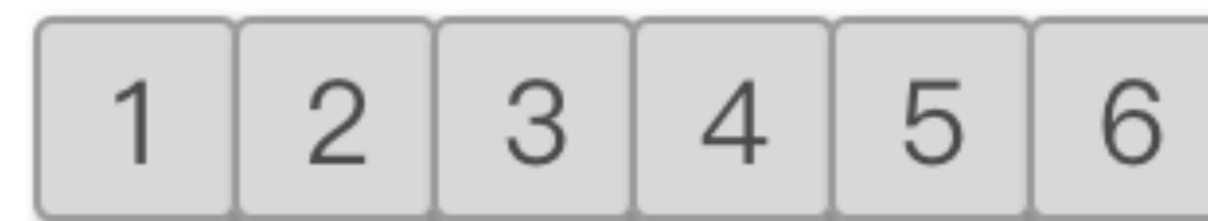


数组

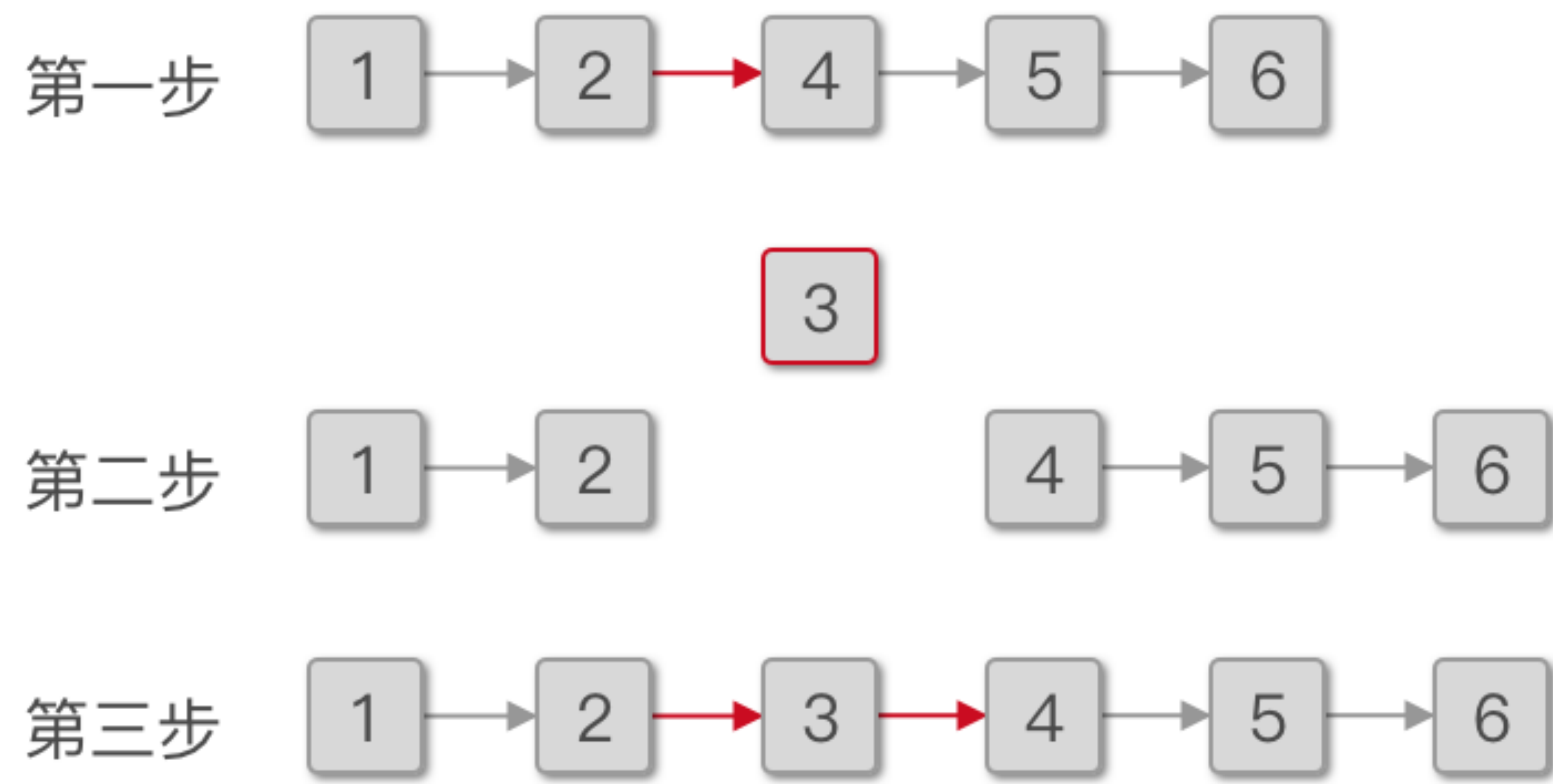
第一步

第二步

第三步

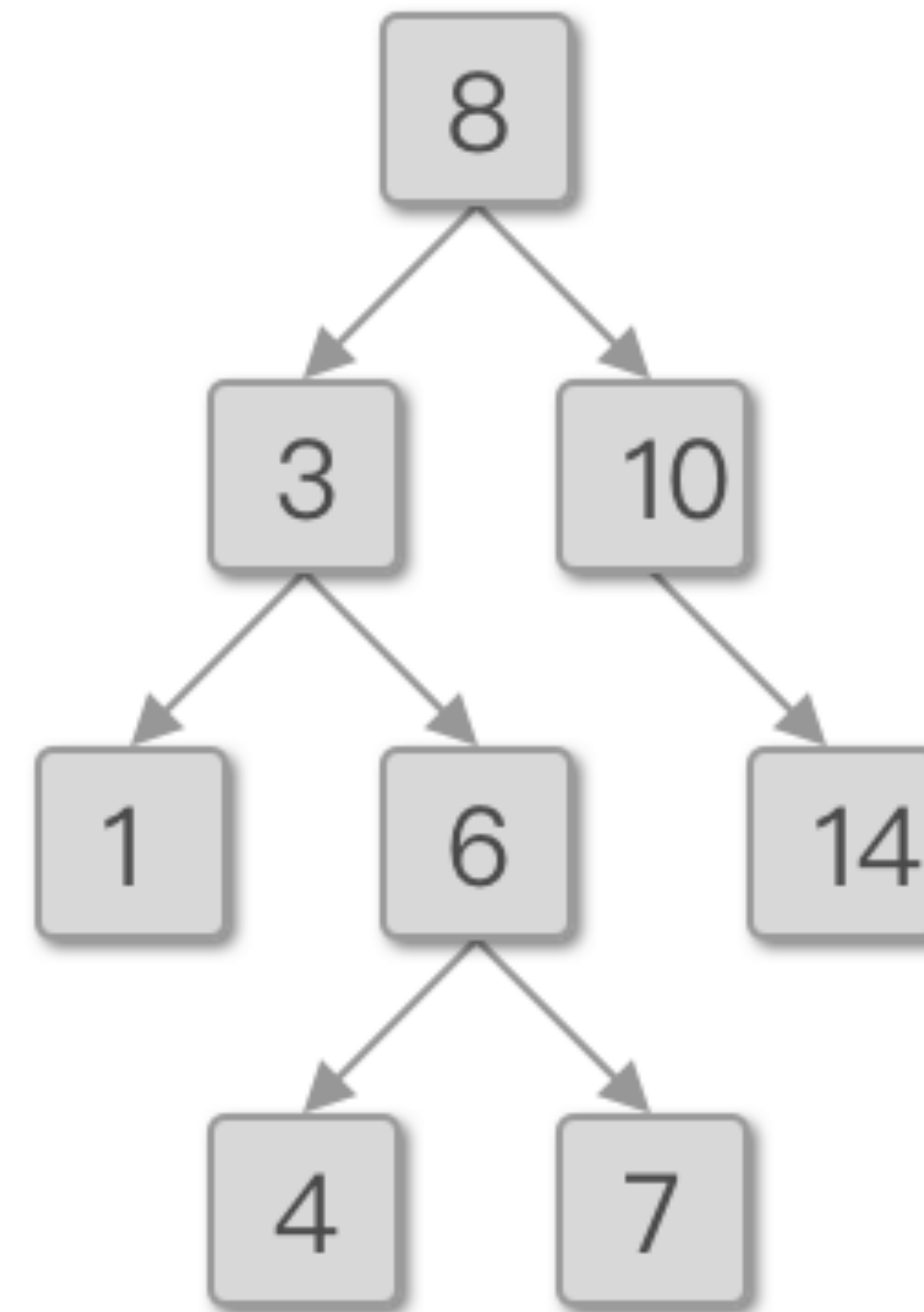


链表



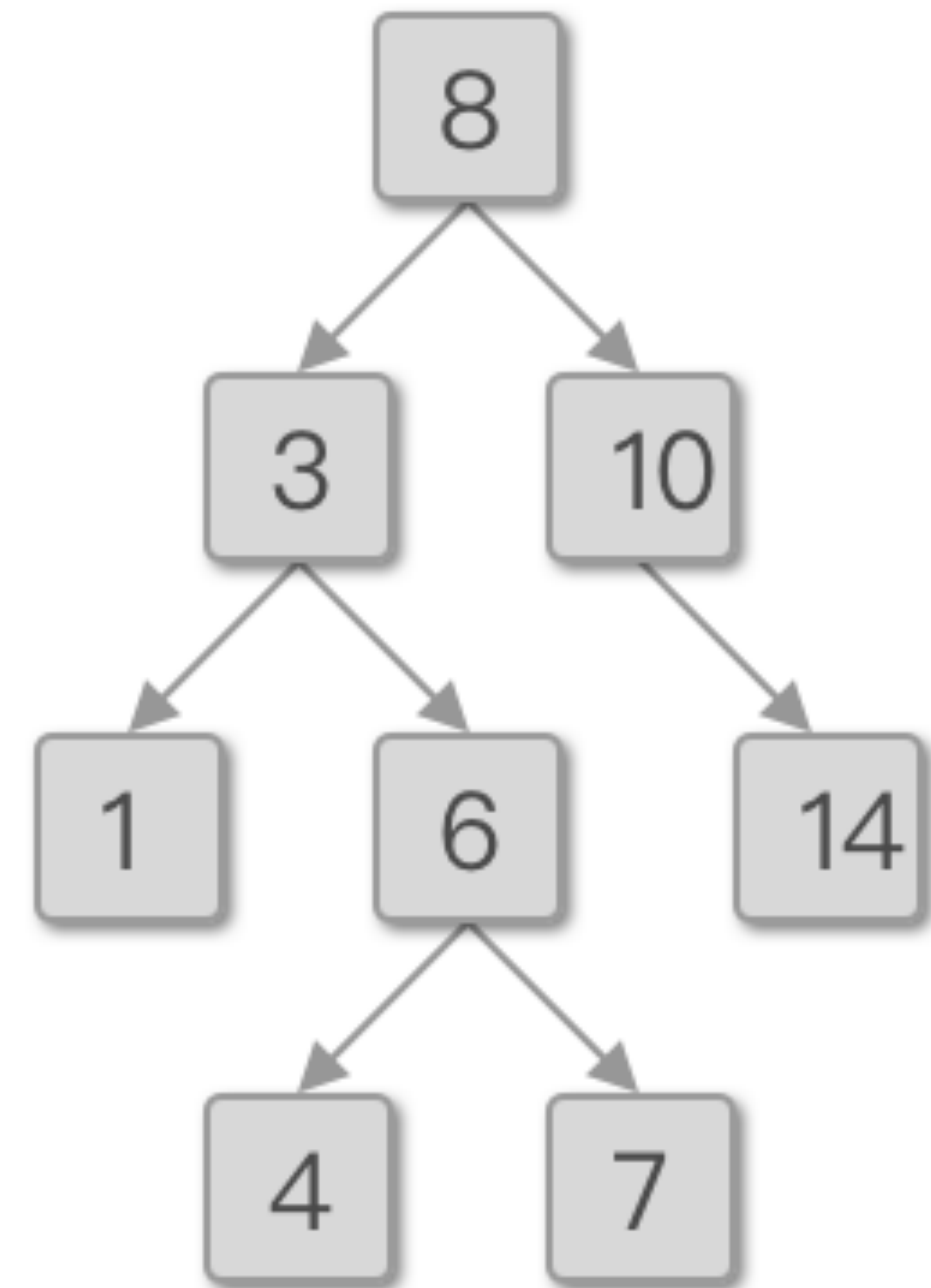
二叉搜索树

- 树的基本术语：
  - 根结点、子节点、父节点、兄弟节点、叶子节点
  - 度：节点的子节点的数量
  - 高度：节点到最远叶子节点的边的数量
  - 深度：节点到根节点的边的数量
  - ……



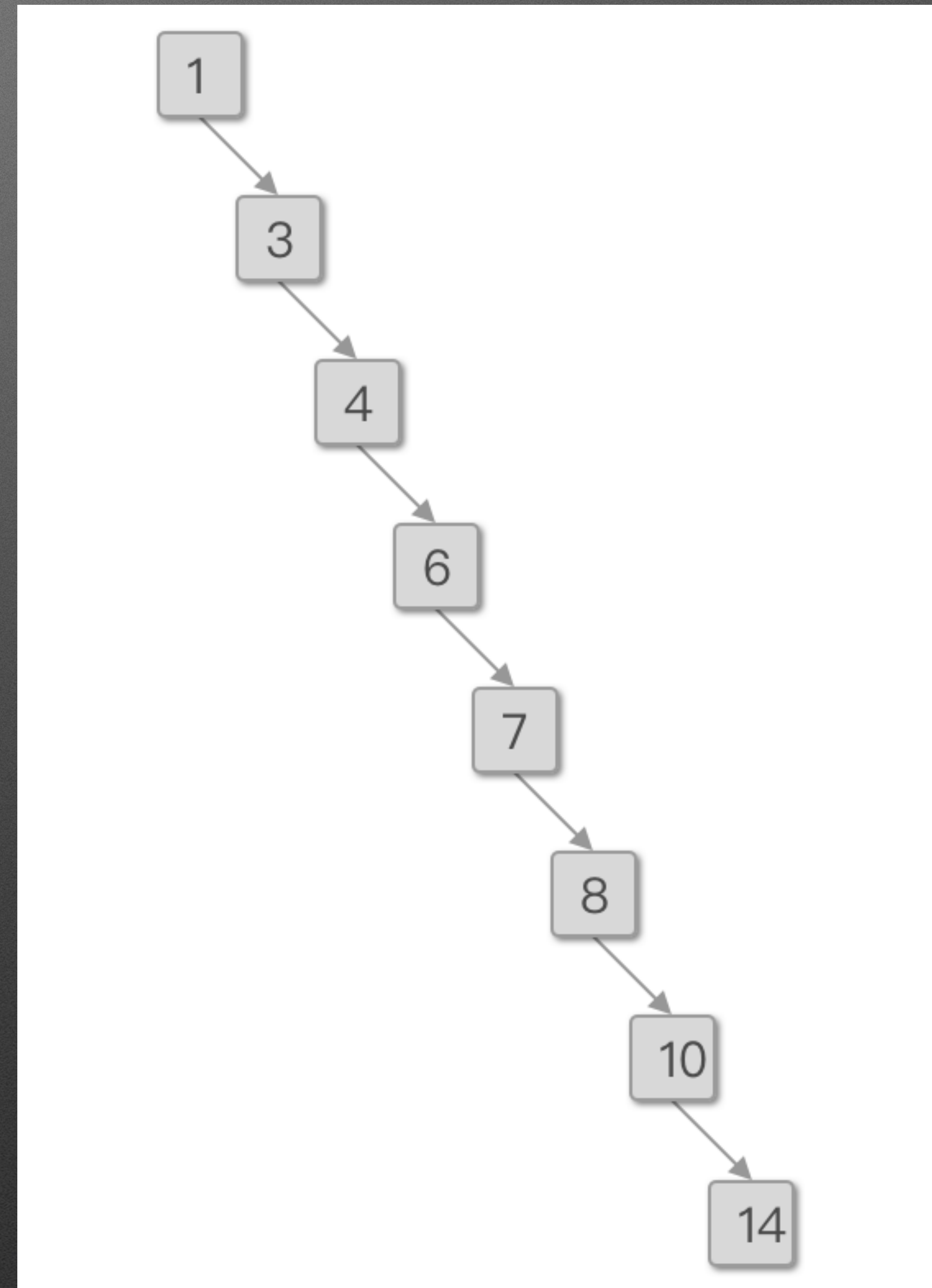
二叉搜索树

- 二叉树：
  i. 每个节点最多只有两个子节点
- 二叉搜索树：
  i. 左子节点 < 当前节点 ≤ 右子节点



二叉搜索树

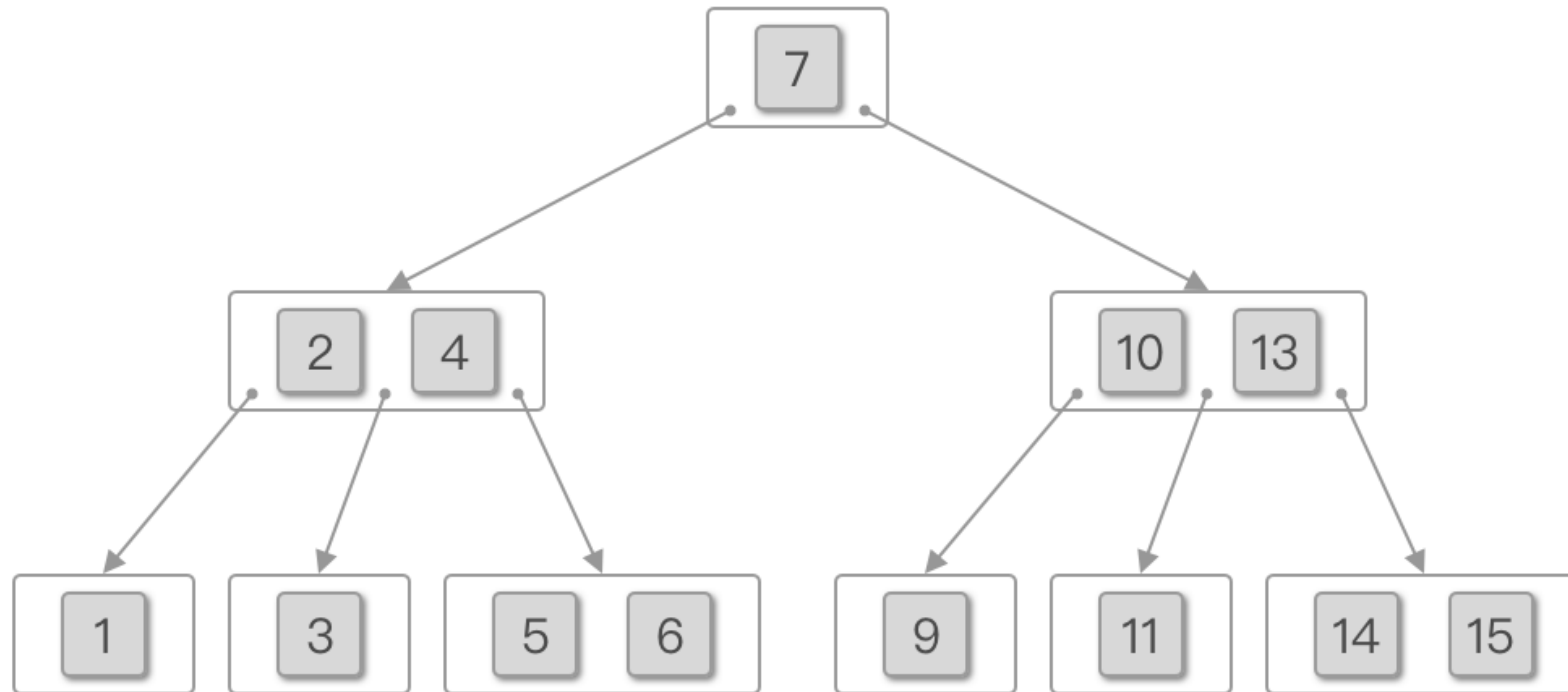
- 二叉树：
  i. 每个节点最多只有两个子节点
- 二叉搜索树：
  i. 左子节点 < 当前节点 ≤ 右子节点



B-Tree

- 在计算机科学中，B-Tree 是一种自平衡的 Tree 数据结构，能保持数据有序性，并且允许查询、有序访问、插入、删除操作在对数时间内完成。
- B-Tree 是 Binary Search Tree 的演化版，它的节点可以包含两个以上的子节点。不同于自平衡的 Binary Search Tree，B-Tree 对大块数据的读写部分进行了优化。

B-Tree



B-Tree

- B-Tree 的内部节点可以包含多个 key，key 用作为分离内部节点的「子树」的分离值。
- B-Tree 预定义了一个「树的度」的范围。当在节点中插入或删除数据时，「节点的度」就会变化，但仅限制在 B-Tree 预定义的范围之内变化。
- 为了维持预定义的「树的度」的范围，B-Tree 的内部节点可能被合并或分离。
- B-Tree 通过要求所有的「叶子节点」都保持相同的深度，来维持树的平衡。

B-Tree 插入元素

www.cs.usfca.edu/~galles/visualization/BTree.html

B-Trees

Insert Delete Find Print Clear

- Max. Degree = 3
- Max. Degree = 4
- Max. Degree = 5
- Max. Degree = 6
- Max. Degree = 7

Preemtive Split / Merge (Even max degree only)

Animation Completed

Skip Back Step Back Pause Step Forward Skip Forward w: 1000 h: 500 Change Canvas Size Move Controls

Animation Speed

Algorithm Visualizations

B-Tree 删除元素

B-Tree 查找元素

www.cs.usfca.edu/~galles/visualization/BTree.html

B-Trees

Insert | Delete | Find | Print | Clear

- Max. Degree = 3
- Max. Degree = 4
- Max. Degree = 5
- Max. Degree = 6
- Max. Degree = 7

Preemtive Split / Merge (Even max degree only)

0008

0004 0012

0002 0006 0010 0014

0001 0003 0005 0007 0009 0011 0013 0015

Animation Completed

Skip Back | Step Back | Pause | Step Forward | Skip Forward | w: 1000 h: 500 | Change Canvas Size | Move Controls

Animation Speed

Algorithm Visualizations

B-Tree 应用场景

- B-Tree 索引适用于全键查找、键值范围查找、键前缀查找：
  - 全值匹配 … WHERE name = ? AND age = ?
  - 匹配最左前缀 … WHERE name = ?
  - 匹配列前缀 … WHERE name LIKE 'A%'
  - 匹配范围值 … WHERE name BETWEEN 'Annie' AND 'Tom'
  - 精确匹配某一列并范围匹配另一列 … WHERE name = ? AND age > 18
  - 只访问索引的查询（覆盖索引） SELECT age … WHERE name = ?
- B-Tree 索引中的节点是有序的，所以除了按值查找之外，索引还可以用于查询中的 ORDER BY 操作。

```
mysql> DESC t;
+-------+------------+------+-----+---------+----------------+
| Field | Type       | Null | Key | Default | Extra          |
+-------+------------+------+-----+---------+----------------+
| id    | int(11)    | NO   | PRI | NULL    | auto_increment |
| name  | char(32)   | NO   | MUL |         |                |
| age   | tinyint(4) | NO   |     | 0       |                |
| phone | char(11)   | NO   |     |         |                |
+-------+------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> SHOW INDEX FROM t;
+-------+------------+--------------------+--------------+-------------+-----------+
| Table | Non_unique | Key_name           | Seq_in_index | Column_name | Collation |
+-------+------------+--------------------+--------------+-------------+-----------+
| t     |          0 | PRIMARY            |            1 | id          | A         |
| t     |          1 | idx_name_age_phone |            1 | name        | A         |
| t     |          1 | idx_name_age_phone |            2 | age         | A         |
| t     |          1 | idx_name_age_phone |            3 | phone       | A         |
+-------+------------+--------------------+--------------+-------------+-----------+
4 rows in set (0.01 sec)

mysql> 
```

B-Tree 应用场景

- B-Tree 索引的限制：
  - 如果不是按照索引的最左列开始查找，则无法使用索引；
  - 不能跳过索引中的列；
  - 如果查询中有某个列的范围查询，则其右边所有列都无法使用索引优化；

```
mysql> DESC t;
+-------+------------+------+-----+---------+----------------+
| Field | Type       | Null | Key | Default | Extra          |
+-------+------------+------+-----+---------+----------------+
| id    | int(11)    | NO   | PRI | NULL    | auto_increment |
| name  | char(32)   | NO   | MUL |         |                |
| age   | tinyint(4) | NO   |     | 0       |                |
| phone | char(11)   | NO   |     |         |                |
+-------+------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> SHOW INDEX FROM t;
+-------+------------+--------------------+--------------+-------------+-----------+
| Table | Non_unique | Key_name           | Seq_in_index | Column_name | Collation |
+-------+------------+--------------------+--------------+-------------+-----------+
| t     |          0 | PRIMARY            |            1 | id          | A         |
| t     |          1 | idx_name_age_phone |            1 | name        | A         |
| t     |          1 | idx_name_age_phone |            2 | age         | A         |
| t     |          1 | idx_name_age_phone |            3 | phone       | A         |
+-------+------------+--------------------+--------------+-------------+-----------+
4 rows in set (0.01 sec)

mysql>
```

B-Tree 的案例分析

- 独立的列
- 索引选择性和前缀索引
- 多列索引、选择合适的索引列顺序
- 冗余和重复的索引
- 聚簇索引
- 覆盖索引
- 使用索引扫描来做排序

B-Tree 的案例分析 · 独立的列

```
mysql> DESC t;
+-------+------------+------+-----+---------+----------------+
| Field | Type       | Null | Key | Default | Extra          |
+-------+------------+------+-----+---------+----------------+
| id    | int(11)    | NO   | PRI | NULL    | auto_increment |
| name  | char(32)   | NO   |     |         |                |
| age   | tinyint(4) | NO   | MUL | 0       |                |
| phone | char(11)   | NO   |     |         |                |
+-------+------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> SHOW INDEX FROM t;
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| t     |          0 | PRIMARY  |            1 | id          | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
| t     |          1 | idx_age  |            1 | age         | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
2 rows in set (0.01 sec)

mysql> EXPLAIN SELECT * FROM t WHERE age = 19;
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t     | NULL       | ref  | idx_age       | idx_age | 1       | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.01 sec)

mysql> EXPLAIN SELECT * FROM t WHERE age + 1 = 19;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | t     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    2 |   100.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql>
```

B-Tree 的案例分析 · 索引选择性

- 索引的选择性是指，不重复的索引值和数据表的记录总数的比值。
- 索引的选择性越高则查询效率越高，因为选择性高的索引可以让 MySQL 在查找时过滤掉更多的行。

B-Tree 的案例分析 · 前缀索引

- 对于 BLOB、TXT、很长的 VARCHAR 类型的列，必须使用前缀索引，因为 MySQL 不允许索引这些列的完整长度。
- 诀窍在于要选择足够长的前缀以保证较高的选择性，同时又不能太长。前缀应该足够长，以使得前缀索引的选择性接近于索引整个列。

Column Prefix Key Parts

For string columns, indexes can be created that use only the leading part of column values, using `col_name(length)` syntax to specify an index prefix length:

- Prefixes can be specified for `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` key parts.
- Prefixes *must* be specified for `BLOB` and `TEXT` key parts. Additionally, `BLOB` and `TEXT` columns can be indexed only for `InnoDB`, `MyISAM`, and `BLACKHOLE` tables.
- Prefix *limits* are measured in bytes. However, prefix *lengths* for index specifications in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements are interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

  Prefix support and lengths of prefixes (where supported) are storage engine dependent. For example, a prefix can be up to 767 bytes long for `InnoDB` tables that use the `REDUNDANT` or `COMPACT` row format. The prefix length limit is 3072 bytes for `InnoDB` tables that use the `DYNAMIC` or `COMPRESSED` row format. For `MyISAM` tables, the prefix length limit is 1000 bytes. The `NDB` storage engine does not support prefixes (see Section 22.1.7.6, "Unsupported or Missing Features in NDB Cluster").

If a specified index prefix exceeds the maximum column data type size, `CREATE INDEX` handles the index as follows:

- For a nonunique index, either an error occurs (if strict SQL mode is enabled), or the index length is reduced to lie within the maximum column data type size and a warning is produced (if strict SQL mode is not enabled).
- For a unique index, an error occurs regardless of SQL mode because reducing the index length might enable insertion of nonunique entries that do not meet the specified uniqueness requirement.

The statement shown here creates an index using the first 10 characters of the `name` column (assuming that `name` has a nonbinary string type):

```
1  CREATE INDEX part_of_name ON customer (name(10));
```

B-Tree 的案例分析 · 多列索引

```
mysql> DESC t;
+-------+------------+------+-----+---------+----------------+
| Field | Type       | Null | Key | Default | Extra          |
+-------+------------+------+-----+---------+----------------+
| id    | int(11)    | NO   | PRI | NULL    | auto_increment |
| name  | char(32)   | NO   | MUL |         |                |
| age   | tinyint(4) | NO   |     | 0       |                |
| phone | char(11)   | NO   |     |         |                |
+-------+------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> SHOW INDEX FROM t;
+-------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| Table | Non_unique | Key_name     | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible |
+-------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| t     |          0 | PRIMARY      |            1 | id          | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
| t     |          1 | idx_name_age |            1 | name        | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
| t     |          1 | idx_name_age |            2 | age         | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
+-------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
3 rows in set (0.00 sec)

mysql> EXPLAIN SELECT * FROM t WHERE name = 'Tom' AND age = 19;
+----+-------------+-------+------------+------+---------------+--------------+---------+-------------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key          | key_len | ref         | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+--------------+---------+-------------+------+----------+-------+
|  1 | SIMPLE      | t     | NULL       | ref  | idx_name_age  | idx_name_age | 97      | const,const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+--------------+---------+-------------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM t WHERE name = 'Tom';
+----+-------------+-------+------------+------+---------------+--------------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key          | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+--------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t     | NULL       | ref  | idx_name_age  | idx_name_age | 96      | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+--------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM t WHERE age = 19;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | t     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    2 |    50.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql>
```

B-Tree 的案例分析 · 冗余的索引

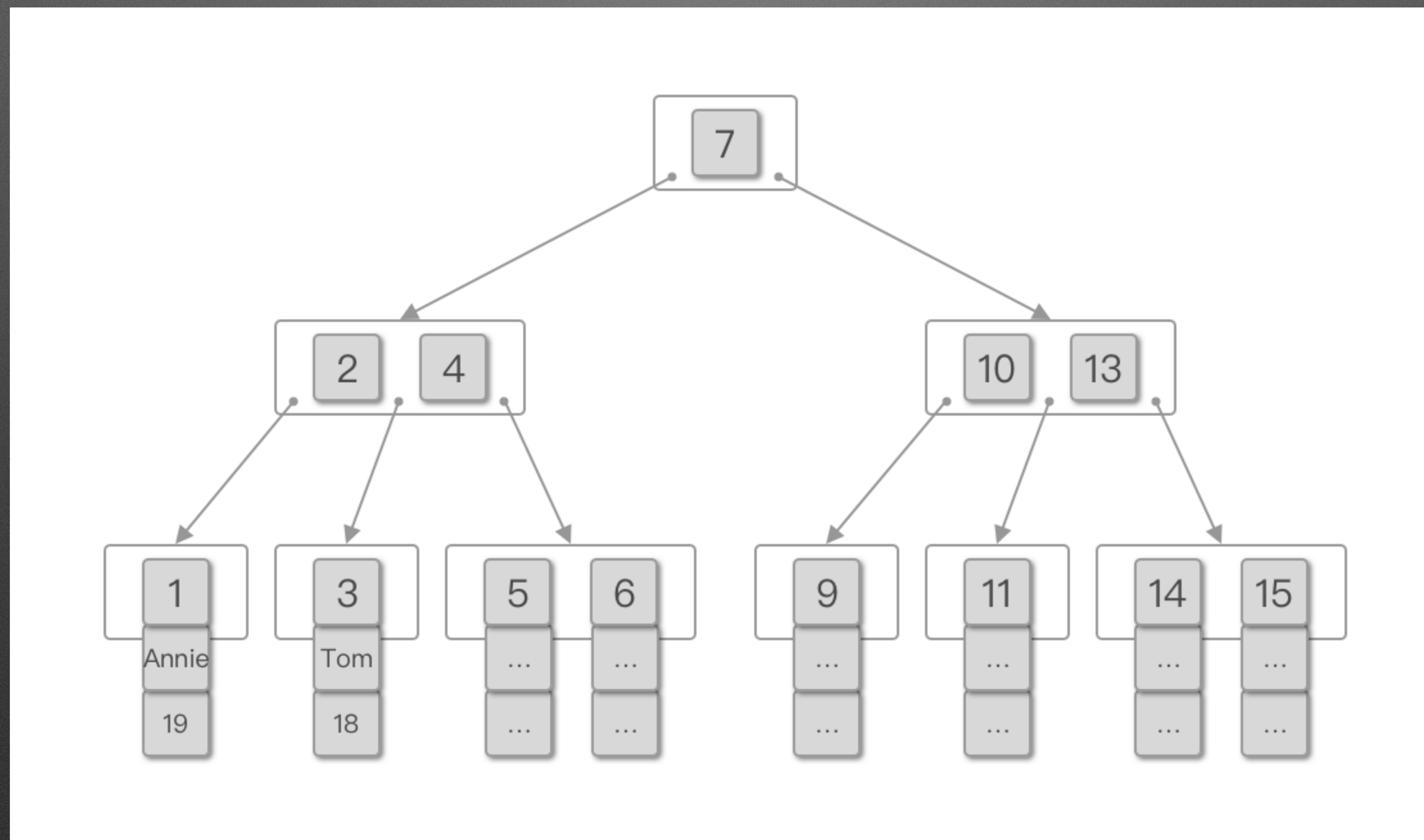
```
mysql> DESC t;
+-------+------------+------+-----+---------+----------------+
| Field | Type       | Null | Key | Default | Extra          |
+-------+------------+------+-----+---------+----------------+
| id    | int(11)    | NO   | PRI | NULL    | auto_increment |
| name  | char(32)   | NO   | MUL |         |                |
| age   | tinyint(4) | NO   |     | 0       |                |
| phone | char(11)   | NO   |     |         |                |
+-------+------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> SHOW INDEX FROM t;
+-------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| Table | Non_unique | Key_name     | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible |
+-------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| t     |          0 | PRIMARY      |            1 | id          | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
| t     |          1 | idx_name_age |            1 | name        | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
| t     |          1 | idx_name_age |            2 | age         | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
| t     |          1 | idx_name     |            1 | name        | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
+-------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
4 rows in set (0.00 sec)

mysql> EXPLAIN SELECT * FROM t WHERE name = 'Tom';
+----+-------------+-------+------------+------+-----------------------+--------------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys         | key          | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+-----------------------+--------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t     | NULL       | ref  | idx_name_age,idx_name | idx_name_age | 96      | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+-----------------------+--------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql>
```

B-Tree 的案例分析 · 聚簇索引



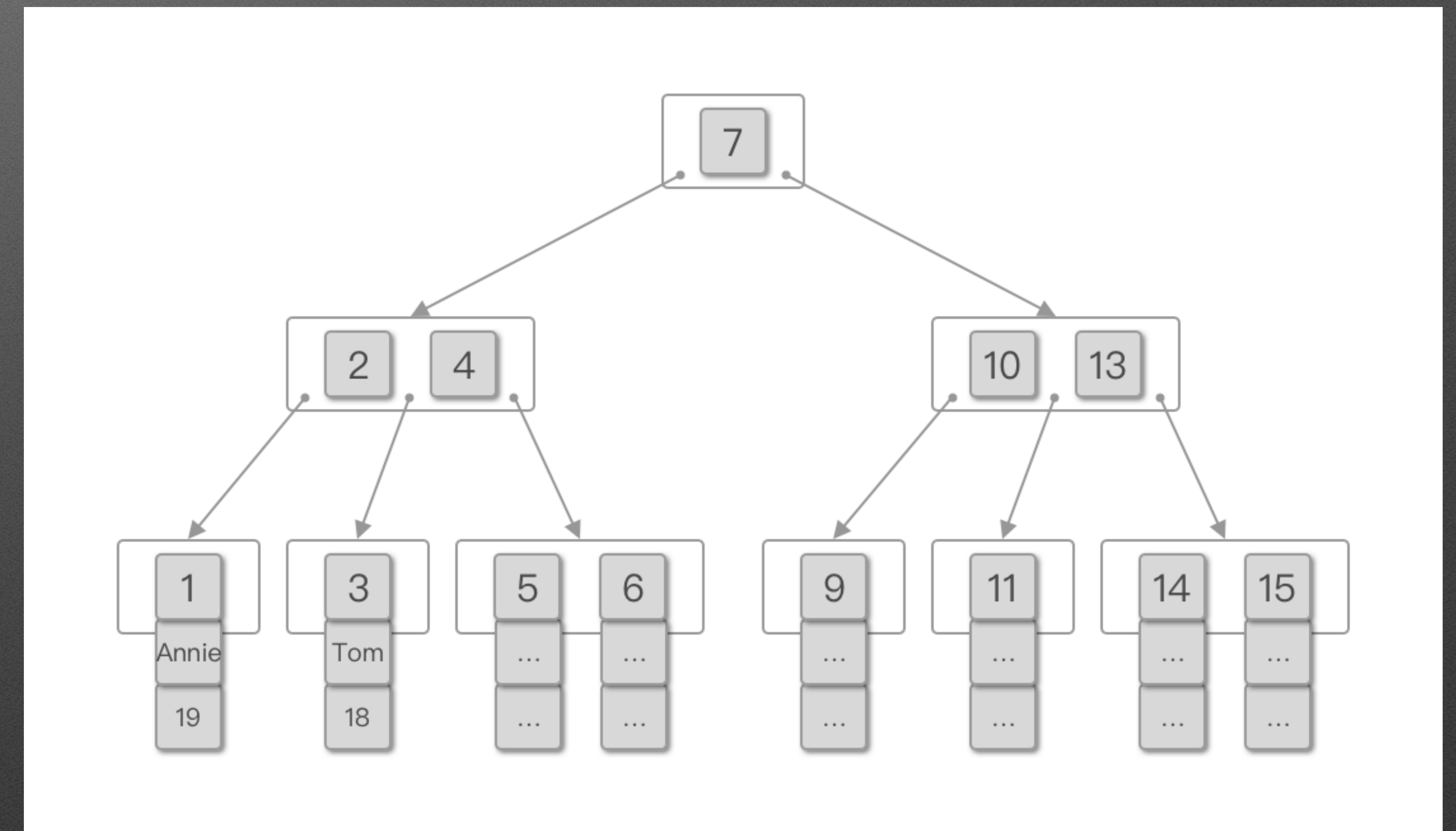
B-Tree 的案例分析 · 聚簇索引

- 聚簇索引不是一种单独的索引，而是一种数据存储的方式。

- InnoDB 的聚簇索引实际上在同一个结构中保存了 B-Tree 索引和数据行。当表有聚簇索引时，它的数据行实际上存放在索引的叶子节点。术语「聚簇」表示数据行和相邻的键值紧凑地存储在一起。

- InnoDB 将通过主键聚集数据，如果没有定义主键，InnoDB 会选择一个唯一的非空索引代替。如果没有这样的索引，InnoDB 会隐式定义一个主键来作为聚簇索引。

- 二级索引访问需要两次索引查找，而不是一次。



B-Tree 的案例分析 · 覆盖索引

```
mysql> DESC t;
+-------+------------+------+-----+---------+----------------+
| Field | Type       | Null | Key | Default | Extra          |
+-------+------------+------+-----+---------+----------------+
| id    | int(11)    | NO   | PRI | NULL    | auto_increment |
| name  | char(32)   | NO   | MUL |         |                |
| age   | tinyint(4) | NO   |     | 0       |                |
| phone | char(11)   | NO   |     |         |                |
+-------+------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> SHOW INDEX FROM t;
+-------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| Table | Non_unique | Key_name     | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible |
+-------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| t     |          0 | PRIMARY      |            1 | id          | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
| t     |          1 | idx_name_age |            1 | name        | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
| t     |          1 | idx_name_age |            2 | age         | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
+-------+------------+--------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
3 rows in set (0.00 sec)

mysql> EXPLAIN SELECT * FROM t WHERE name = 'Tom';
+----+-------------+-------+------------+------+---------------+--------------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key          | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+--------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t     | NULL       | ref  | idx_name_age  | idx_name_age | 96      | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+--------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT age FROM t WHERE name = 'Tom';
+----+-------------+-------+------------+------+---------------+--------------+---------+-------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key          | key_len | ref   | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+--------------+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | t     | NULL       | ref  | idx_name_age  | idx_name_age | 96      | const |    1 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+--------------+---------+-------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql>
```

B-Tree 的案例分析 · 覆盖索引

- 如果一个索引包含（覆盖）所有需要查询的字段的值，我们就称之为覆盖索引。覆盖索引是非常有用的工具，能够极大地提高性能：

  i. 索引条目通常远小于数据行的大小，所以如果只需要读取索引，那 MySQL 就会极大地减少数据访问量；

  ii. 因为索引是按照列值顺序存储的，所以对于 I/O密集型的范围查询会比从磁盘随机读取每一行数据的 I/O 要少得多；

  iii. 由于 InnoDB 使用聚簇索引存储数据，所以如果二级索引能够覆盖查询，则可以避免对主键索引的二次查询；

- MySQL 只能使用 B-Tree 索引做覆盖索引。

- 当发起一个覆盖索引的查询时，在 EXPLAIN 的 Extra 列可以看到 “Using Index” 的信息。

B-Tree 的案例分析 · 索引顺序扫描

```
mysql> DESC t;
+-------+------------+------+-----+---------+----------------+
| Field | Type       | Null | Key | Default | Extra          |
+-------+------------+------+-----+---------+----------------+
| id    | int(11)    | NO   | PRI | NULL    | auto_increment |
| name  | char(32)   | NO   |     |         |                |
| age   | tinyint(4) | NO   | MUL | 0       |                |
| phone | char(11)   | NO   |     |         |                |
+-------+------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> SHOW INDEX FROM t;
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| t     |          0 | PRIMARY  |            1 | id          | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
| t     |          1 | idx_age  |            1 | age         | A         |           2 |     NULL |   NULL |      | BTREE      |         |               | YES     |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
2 rows in set (0.01 sec)

mysql> EXPLAIN SELECT * FROM t ORDER BY age;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
|  1 | SIMPLE      | t     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    2 |   100.00 | Using filesort |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT id FROM t ORDER BY age;
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | t     | NULL       | index | NULL          | idx_age | 1       | NULL |    2 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> DROP INDEX idx_age ON t;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN SELECT id FROM t ORDER BY age;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
|  1 | SIMPLE      | t     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    2 |   100.00 | Using filesort |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
1 row in set, 1 warning (0.00 sec)

mysql>
```

B-Tree 的案例分析 · 索引顺序扫描

- MySQL 有两种方式可以生成有序的结果：通过排序操作、通过索引顺序扫描。如果 EXPLAIN 结果中的 type 列的值为 index，则说明 MySQL 使用索引扫描来做排序。

- 扫描索引本身是很快的，只需要从一条索引记录移动到紧接着的下一条记录。但如果索引不能覆盖查询所需的全部列，那就不得不每扫描一条索引记录都回表查询一次对应的行。因此按照索引顺序读取数据的速度通常要比顺序地按表扫描慢。

- MySQL 可以使用同一个索引既满足排序，又用于查找。

- ORDER BY 子句和 WHERE 子句的索引限制是一样的。

参考资料

- 《高性能 MySQL》
- 《MySQL 技术内幕 - InnoDB 存储引擎》
- MySQL 5.7 Reference Manual
- CodingLabs - MySQL 索引背后的数据结构及算法原理
- 老赵点滴 - 浅谈代码的执行效率：缓存与局部性
- 鳥哥的 Linux 私房菜 - 計算機概論 - 硬碟與儲存設備
- 数据结构可视化 https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

THANKS