数据库高级特性

一、权限管理

MySQL 的权限分为两个阶段：

1. 第一阶段为连接验证，主要限制用户连接 mysql-server 时使用的 ip 及 密码
2. 第二阶段为操作检查，主要检查用户执行的指令是否被允许，一般非管理员账户不被允许执行 drop、delete 等危险操作

权限控制安全准则：

1. 只授予能满足需要的最小权限，防止用户执行危险操作。
2. 限制用户的登录主机，防止不速之客登录数据库。
3. 禁止或删除没有密码的用户。
4. 禁止用户使用弱密码。
5. 定期清理无效的用户，回收权限或者删除用户。

常用操作：

1. 创建账户、权限授予
   - 8.0 之前版本

     ```
     GRANT ALL PRIVILEGES on *.* to '用户名'@'主机' IDENTIFIED BY "密码" WITH
     GRANT OPTION;
     flush privileges; -- 刷新使权限生效
     ```

     - `ALL PRIVILEGES`: 授予全部权限, 也可以指定 `select`、`insert` 等
     - `*.*`: 允许操作的数据库和表
   - `WITH GRANT OPTION`: 带有该子句说明允许用户将自己拥有的权限授予别人
   - 8.0 之后的版本

     ```
     CREATE USER `用户名`@`主机` IDENTIFIED BY '密码';  -- 创建账户
     GRANT ALL ON *.* TO `用户名`@`主机` WITH GRANT OPTION;  -- 授权
     ```

2. 修改密码

   ```
   ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '你
   的密码';"
   ```

3. 查看权限

```
show grants;  -- 查看当前用户的权限
show grants for 'abc'@'localhost';  -- 查看用户 abc 的权限
```

4. 回收权限

```
revoke delete on *.* from 'abc'@'localhost';
```

5. 删除用户

```
use mysql;
select host, user from user;
drop user 用户名@'%';
```

二、视图

- 视图是数据的特定子集，是从其他表里提取出数据而形成的虚拟表，或者说临时表。
- 创建视图表依赖一个查询。
- 视图是永远不会自己消失的除非手动删除它。
- 视图有时会对提高效率有帮助。临时表不会对性能有帮助，是资源消耗者。
- 视图一般随该数据库存放在一起，临时表永远都是在 tempdb 里的。
- 视图适合于多表连接浏览时使用；不适合增、删、改，这样可以提高执行效率。
- 一般视图表的名称以 `v_` 为前缀，用来与正常表进行区分。
- 对原表的修改会影响到视图中的数据。

创建视图

- 语法: `create view 视图名 as 查询语句`
- 示例:

```
-- 以上节课的关联查询为例
create view v_user_score as
select a.id, a.name, b.math, b.english
from student a inner join score b on a.id=b order by id;

-- 查询
select * from v_user_score;

-- 删除
drop view v_user_score;
```

三、存储引擎

存储引擎就是如何存储数据、如何为数据建立索引和如何更新、查询数据等技术的实现方法。

MySQL 默认支持多种存储引擎，以适用于不同领域 的数据库应用需要，用户可以通过选择使用不同的存储引擎提高应用的效率，提供灵活的存储。

查看当前的存储引擎

```
show variables like '%storage_engine';
show engines;
```

MySQL 常用的存储引擎

功 能	MYISAM	Memory	InnoDB	Archive
存储限制	256TB	RAM	64TB	None
支持事物	No	No	Yes	No
支持全文索引	Yes	No	No	No
支持数索引	Yes	Yes	Yes	No
支持哈希索引	No	Yes	No	No
支持数据缓存	No	N/A	Yes	No
支持外键	No	No	Yes	No

<!-- ### 3. 表的引擎

InnoDB 和 MyISAM
CURD操作:增删改查
C create insert 插入
U update 修改
R read select 查询
D delete 删除
less /etc/my.cnf
默认的存储路径
datadir = /data/mysql

innodb 在 写的操作上非常的有优势(事务) CUD全是写的操作
myisam 在 读的操作上非常的有优势(健全的索引) R操作

引擎的存储方式

myisam将一张表存储为三个文件
demo.frm -> 表的结构
demo.MYD -> 存储的是数据
demo.MYI -> 存储的是表的索引

myisam的文件可以任意的移动

innodb将一张表存储为两个文件
demo.frm -> 表的结构+表的索引
demo.ibd -> 存储的是数据
ibd存储是有限的, 存储不足自动创建ibd1, ibd2

innodb的文件创建在哪个数据库中, 不认任意的移动 -->

1. InnoDB

   事务型数据库的首选引擎，支持事务安全表（ACID），支持行锁定和外键，InnoDB是默认的MySQL引擎。

   InnoDB主要特性有：

   1. InnoDB 给 MySQL 提供了具有提交、回滚、崩溃恢复能力的事物安全存储引擎。
   2. InnoDB 是为处理巨大数据量的最大性能设计。它的 CPU 效率比其他基于磁盘的关系型数据库引擎高。
   3. InnoDB 存储引擎自带缓冲池，可以将数据和索引缓存在内存中。
   4. InnoDB 支持外键完整性约束。
   5. InnoDB 被用在众多需要高性能的大型数据库站点上
   6. InnoDB 支持行级锁
2. MyISAM

   MyISAM 基于 ISAM 存储引擎，并对其进行扩展。它是在Web、数据仓储和其他应用环境下最常使用的存储引擎之一。MyISAM 拥有较高的插入、查询速度，但不支持事物。

   MyISAM主要特性有：

   1. 大文件支持更好
   2. 当删除、更新、插入混用时，产生更少碎片。
   3. 每个 MyISAM 表最大索引数是64，这可以通过重新编译来改变。每个索引最大的列数是16
   4. 最大的键长度是1000字节。
   5. BLOB和TEXT列可以被索引
   6. NULL 被允许在索引的列中，这个值占每个键的0~1个字节
   7. 所有数字键值以高字节优先被存储以允许一个更高的索引压缩
   8. MyISAM 类型表的 AUTO_INCREMENT 列更新比 InnoDB 类型的 AUTO_INCREMENT 更快
   9. 可以把数据文件和索引文件放在不同目录
   10. 每个字符列可以有不同的字符集
   11. 有 VARCHAR 的表可以固定或动态记录长度
   12. VARCHAR 和 CHAR 列可以多达 64KB
   13. 只支持表锁
3. MEMORY

   MEMORY 存储引擎将表中的数据存储到内存中，为查询和引用其他表数据提供快速访问。

存储引擎的选择

一般来说，对插入和并发性能要求较高的，或者需要外键及事务支持的选择 InnoDB，

插入较少，查询较多的场景，优先考虑 MyISAM。

使用引擎

一般在建表时添加

```
create table abc (
    name char(10)
) engine=MyISAM charset=utf8;

create table xyz (
    name char(10)
) engine=InnoDB charset=utf8;
```

四、索引

索引就是为特定的 mysql 字段进行一些特定的算法排序，比如二叉树的算法和哈希算法，哈希算法是通过建立特征值，然后根据特征值来快速查找。

MySQL 索引的建立对于 MySQL 的高效运行是很重要的，索引可以大大提高MySQL的检索速度。

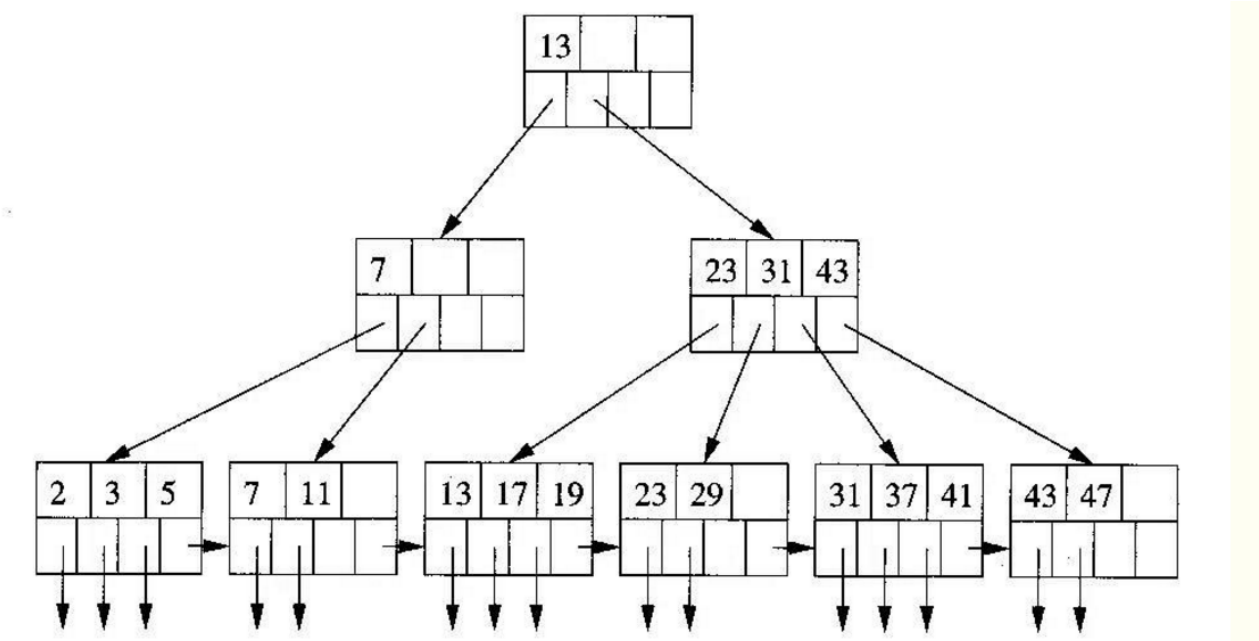
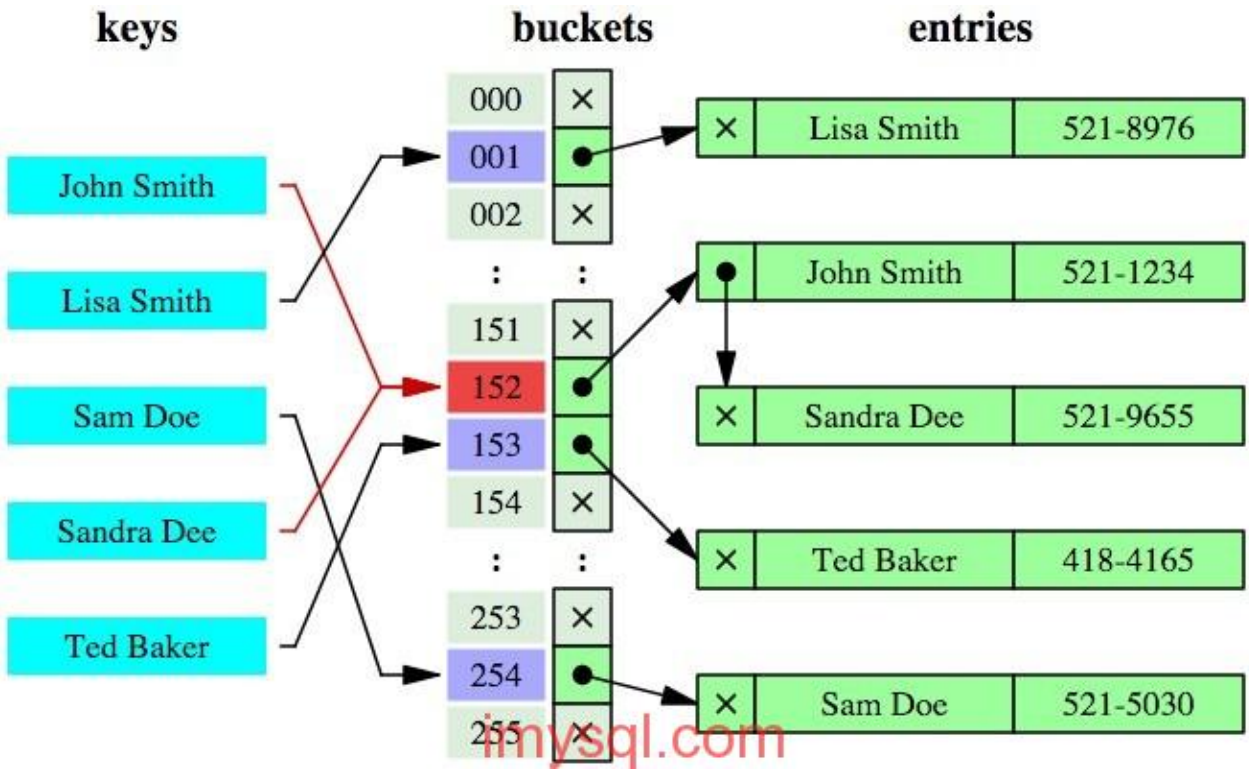


Figure 13.23: A B-tree

用的最多，并且是 mysql 默认的索引数据结构 btree。

通过 BTREE 算法建立索引的字段，比如扫描 20 行就能得到未使用 BTREE 前扫描了 2^20 行的结果。



哈希索引比较特殊，时间复杂度为 O(1), 但只适合等值比较方式的查询，不适合范围或大小比较进行查询

索引的优点:

- 一个字：快！使用索引能极大提升查询速度。

索引的缺点:

1. 额外的使用了一些存储的空间
2. 索引会让写的操作变慢

索引的创建原则

1. 适合用于频繁查找的列
2. 适合经常用于条件判断的列
3. 适合经常由于排序的列
4. 不适合数据不多的列
5. 不适合很少查询的列

创建索引

1. 建表时添加索引

```
create table 表(
    id int not null,
    username varchar(16) not null,
    index 索引名(字段名(长度))
);
```

2. 后期添加索引

```
create index `索引名` on 表名(字段名(长度));
```

删除索引

```
drop index [索引名] on 表;
```

唯一索引

它与前面的普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一。

```
create unique index 索引名 on 表(字段名(长度));

-- 或
create table 表(
    id int not null,
    username varchar(16) not null,
    unique 索引名 (字段名(长度))
);
```

查看索引

```
show index from table_name;
```

五、关系与外键

关系

- 一对一
  - 在 A 表中有一条记录，在 B 表中同样有唯一条记录相匹配
  - 比如: 学生表和成绩表
- 一对多
  - 在 A 表中有一条记录，在 B 表中有多条记录一直对应
  - 比如: 电商中的用户表与订单表
- 多对多
  - A 表中的一条记录有多条 B 表数据对应, 同样 B 表中一条数据在 A 表中也有多条与之对应
  - 比如: 用户与权限关系

外键

外键是一种约束。他只是保证数据的一致性，并不能给系统性能带来任何好处。

建立外键时，都会在外键列上建立对应的索引。外键的存在会在每一次数据插入、修改时进行约束检查，如果不满足外键约束，则禁止数据的插入或修改，这必然带来一个问题，就是在数据量特别大的情况下，每一次约束检查必然导致性能的下降。

出于性能的考虑，如果我们的系统对性能要求较高，那么可以考虑在生产环境中不使用外键。

1. 构造数据

```
-- 用户表
create table `user` (
    `id` int unsigned primary key auto_increment,
    `name` char(32) not null unique
) charset=utf8;

-- 商品表
create table `product` (
    `id` int unsigned primary key auto_increment,
    `name` char(32) not null unique,
    `price` float
) charset=utf8;

-- 用户信息表：一对一
create table `userinfo` (
    `id` int unsigned primary key auto_increment,
    `phone` int unsigned unique,
    `age` int unsigned,
    `location` varchar(128)
) charset=utf8;

-- 用户组表：一对多
create table `group` (
    `id` int unsigned primary key auto_increment,
    `name` char(32) not null unique
) charset=utf8;

-- 订单表：多对多
create table `order` (
    `id` int unsigned primary key auto_increment,
    `uid` int unsigned,
    `pid` int unsigned
) charset=utf8;
```

2. 添加外键

```
-- 为 user 和 userinfo 建立关联的外键
alter table userinfo add constraint fk_user_id foreign key(id) references
user(id);

-- 建立用户与组的外键约束
alter table `user` add `gid` int unsigned;
alter table `user` add constraint `fk_group_id` foreign key(`gid`)
references `group`(`id`);

-- 建立用户、商品、订单的外键约束
alter table `order` add constraint `fk_user_id` foreign key(`uid`)
references `user`(`id`);
alter table `order` add constraint `fk_prod_id` foreign key(`pid`)
references `product`(`id`);
```

3. 尝试插入数据后在删除，分别先对主表和子表进行一次删除