

必备的数据库基础训练

授课人 令狐东邪

版权声明

九章的所有课程均受法律保护，不允许录像与传播录像
一经发现，将被追究法律责任和赔偿经济损失

数据库 Database

大多数人或多或少都有听说过或者了解过数据库，但是可能很少有人真正清楚什么是数据库，因为数据库这个术语常常被人混用。比如有人可能会问你，“你用过什么数据库啊？”。你可能会很自然的回答，“我用过 MySQL，或者 PostgreSQL 等等”。实际上 MySQL 是一个数据库软件，我们称之为**数据库管理系统 (DBMS)**。而真正的**数据库**其实是一个以某种结构存储数据的**容器**，我们通过 DBMS 去创建和操作数据库。



表 Table

如果我们把数据库比作一个仓库，把数据比作不同的货物，那么货物在仓库里如何存放，是堆叠还是平铺，如何分类等等都是需要考虑。如果我们把货物一股脑地堆成一堆，那么要去取货物的时候就会十分麻烦。因此我们通常会把不同的货物分类放到不同的区域，根据不同的货物采用不同的堆放方式，这就是**表**的概念。我们在存储数据时，不会直接把数据存进数据库，而是会先创建不同的表，然后把数据放进不同的表里。比如我们会把用户信息放进用户表中，推特信息放进推特表中。

User Table

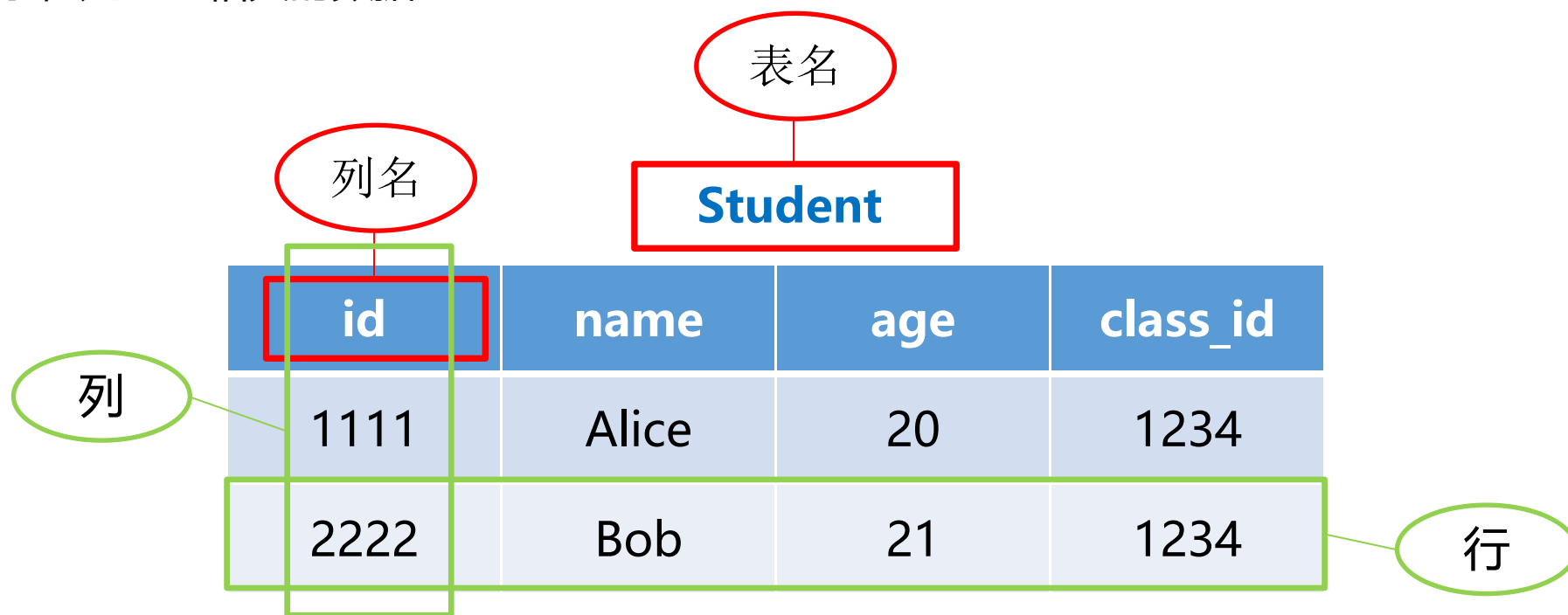
id	name	password
1	张三	*****
2	李四	*****
...

列 Column

表中的一个字段，所有表都是由一个或多个列组成的。一列包含了相同类型的数据。

行 Row

表中的一个记录，是一组相关的数据。



id	name	age	class_id
1111	Alice	20	1234
2222	Bob	21	1234

什么是 SQL

SQL 是 **Structured Query Language**（结构化查询语言）的缩写。SQL 是一种专门用来与数据库沟通的语言，用于存储数据以及查询、更新和管理关系数据库系统。

优点:

- 几乎所有重要的 DBMS 都支持 SQL，所以学习了 SQL，你几乎能与所有数据库打交道。
- SQL 简单易学。它的语句全都有很强的描述性，而且单词的数目不多。
- SQL 可以进行非常复杂和高级的数据库操作。

Tweet Table

字段 field	id	user_id	content	created_at	likes_count	comments_count
描述	id	用户 id	推特内容	创建时间	点赞数	评论数
类型	INT	INT	LONGTEXT	DATETIME	BIGINT	BIGINT

```
class Tweet(models.Model):
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    photos = models.ManyToManyField(Photo, blank=True)
    ...
    likes_count = models.BigIntegerField(default=0, null=True)
    comments_count = models.BigIntegerField(default=0, null=True)
```

数据库表单 vs Django Model

Tweet Table

字段 field	id	user_id	content	created_at	likes_count	comments_count
描述	id	用户 id	推特内容	创建时间	点赞数	评论数
类型	INT	INT	LONGTEXT	DATETIME	BIGINT	BIGINT

数据类型

在上面的表单中，我们用到了4种数据类型，不同的数据类型定义了列中可以存储什么样的数据。

数据类型	大小	范围	作用
INT	4 bytes	(-2 147 483 648, 2 147 483 647)	大整数值
BIGINT	8 bytes	(-9,223,372,036,854,775,808, 9 223 372 036 854 775 807)	极大整数值
LONGTEXT	0-4 294 967 295 bytes		极大文本数据
DATETIME	8 bytes	1000-01-01 00:00:00/9999-12-31 23:59:59	日期和时间值

Tweet Table

字段 field	id	user_id	content	created_at	likes_count	comments_count
描述	id	用户 id	推特内容	创建时间	点赞数	评论数
类型	INT	INT	LONGTEXT	DATETIME	BIGINT	BIGINT
约束	主键	外键	非空			

约束

主键约束 (PRIMARY KEY)

每张表都会有一个主键，主键的作用是唯一标识表中的一行数据，也就是说一张表里不可能存在主键值相同的两行记录，并且主键不能是 null 值，每个表里最多只能有一个主键。

非空约束 (NOT NULL)

如果指定了某个为非空约束，则在插入数据时这个字段不能为空，否则数据库会报错。

Tweet Table

字段 field	id	user_id	content	created_at	likes_count	comments_count
描述	id	用户 id	推特内容	创建时间	点赞数	评论数
类型	INT	INT	LONGTEXT	DATETIME	BIGINT	BIGINT
约束	主键	外键	非空			

User Table

字段 field	id	user_name	password
描述	id	用户名	密码
类型	INT	VARCHAR	VARCHAR

数据类型

在用户表中，我们又认识了一种新的数据类型，字符串类型。

数据类型	大小	作用
VARCHAR	0-65535 bytes	变长字符串

Tweet Table

字段 field	id	user_id	content	created_at	likes_count	comments_count
描述	id	用户 id	推特内容	创建时间	点赞数	评论数
类型	INT	INT	LONGTEXT	DATETIME	BIGINT	BIGINT
约束	主键	外键	非空			

User Table

字段 field	id	user_name	password
描述	id	用户名	密码
类型	INT	VARCHAR	VARCHAR
约束	主键	非空	非空

约束

外键约束 (FOREIGN KEY)

外键的作用是关联两个表，比如这里的推特表和用户表，外键所在的表为从表，外键所指向的表为主表。推特表中的 user_id 实际上就是用户表的 id。

外键插入

插入准则

在tweet表中插入数据的user_id必须对应user表中id的某个值

阻止执行

从表插入新行，其外键值不是主表的主键值便阻止插入；

从表修改外键值，新值不是主表的主键值便阻止修改；

主表删除行，其主键值在从表里存在便阻止删除(要想删除，必须先删除从表的相关行)；

主表修改主键值，旧值在从表里存在便阻止修改(要想修改，必须先删除从表的相关行)。

级联执行

主表删除行，连带从表的相关行一起删除；

主表修改主键值，连带从表相关行的外键值一起修改。

使用外键的建议

外键是把双刃剑，为我们带来便利的同时，也引入了一些问题，让我们一起来看看吧



优点

1. 由数据库自身帮我们维护约束，可以迅速的建立一个可靠性非常高数据库，为我们快速搭建系统提供了基础
2. 使用外键有更好的描述性，可以通过观察外键了解业务逻辑，会使设计周到具体全面。



缺点

1. 性能问题，数据库维护外键约束，会消耗很多的资源，导致数据库负载变高，反应变慢
2. 表之间耦合度变高，导致系统升级困难

Tweet Table

id	user_id	content	created_at_date	likes_count	comments_count
1	1	真的直接泪奔	2021-04-01	234	342
2	2	真诚打动内心	2021-04-01	11	4
3	2	妈妈远比你想象中更爱更爱你	2021-04-02	546	34
4	1	我爱你	2021-04-03	324	234

User Table

id	user_name	password
1	张伟	fme+ftwl-&o!
2	王五	fsymce2b%rle

多对一 (ManyToOne) 关系

外键为我们“多对一”的需求提供了技术基础，在从表 (UserTable) 中创建一个“user_id”字段对应主表 (Tweet Table) 的“id”字段。Tweet Table 的“user_id”是可重复的，但是UserTable的“id”字段是不重复、非空的。

多对多 (ManyToMany) 关系

通过设计中间表的方式，
我们可以为数据建立多对
多关系

Tweet Table

id	user_id	content	created_at_date	likes_count	comments_c ount
1	1	真的直接泪奔	2021-04-01	234	342
2	1	真诚打动内心	2021-04-01	11	4

Tweet_Photo Table

id	tweet_id	photo_id
1	1	1
2	2	1
3	2	2
4	2	3

Photo Table

id	status	file	has_deleted	deleted_at	created_at	user_id
1	0	/img/1.jpg	0	null	2021-04-06 09:24	1
2	2	/img/2.jpg	0	null	2021-04-06 09:24	1
3	2	/img/3.jpg	0	null	2021-04-06 09:24	1

前面介绍了数据库的一些概念，现在让我们来实操一下吧。

插入 INSERT

小王报名了九章算法的《Twitter 后端系统 - Django 项目实战》课程，想通过写推特的方式记录下自己学习的历程，于是她准备发一条推特纪念一下今天的日子。

努力的人不会输



小王的推文包含了一段文字和两张图片。

这段文字和图片会发送到推特的服务器，并将文本和图片保存的服务器上

Tips: 文本会插入推文表 (Tweet Table)，图片信息保存在图片表 (Photo Table)，图片文件会保存在文件系统 (File System) 中。

插入 INSERT

通过执行如下语句，我们可以在 Tweet 表中插入一行新的记录：

```
INSERT INTO tweet ( user_id, content, created_at, likes_count, comments_count )  
VALUES( 1, "努力的人不会输", "2021-04-06 12:00", 0, 0 )
```

 SQL

```
Tweet.objects.create(user=user, content='努力的人不会输')
```

 Django ORM

INSERT INTO 关键字表示这是一个插入数据的操作，tweet 表示表名，紧跟其后的小括号中的字段表示要插入数据的字段，**VALUES** 表示后面括号中要放实际的数据，并且数据的顺序要和前面字段的顺序保持一致。

表中的id值通常会由数据库自动编号。

Tweet Table

id	user_id	content	created_at	likes_count	comments_count
1	1	努力的人不会输	2021-04-06 12:00	0	0

插入 INSERT

因为一个推文可以包含多张图片，并且图片本身又有一些属性 (创建时间、是否删除、删除时间)，因此图片信息单独保存在一张表中。

```
INSERT INTO photos_photo ( `status`, `file`, `has_deleted`, `deleted_at`, `created_at`, `created_at_date`,  
`user_id` )  
VALUES ( 0, '1.jpg', 0, NULL, '2021-04-06 12:00', '2021-04-06', 1 ),  
        ( 0, '2.jpg', 0, NULL, '2021-04-06 12:00', '2021-04-06', 1 )
```

可以在一条sql语句中插入多条记录，括号之间使用英文逗号分隔。

Photo Table

id	status	file	has_deleted	deleted_at	created_at	user_id
1	0	/img/1.jpg	0	null	2021-04-06 12:00	1
2	2	/img/2.jpg	0	null	2021-04-06 12:00	1

查找 SELECT

经过一段时间的学习，小王每天都在推特上打卡记录自己的学习日志，小王想回顾一下自己这些天的学习路程，可以使用如下语句查询：



```
SELECT * FROM tweet WHERE user_id=xx; SQL
```

```
queryset = Tweet.objects.filter(user_id=user_id) Django ORM
```

我们可以使用 **SELECT** 关键字来进行查询，***** 表示查询表中的所有字段，**FROM** 表示要查询的表是 tweets 表。我们可以使用 **WHERE** 关键字来指定查询条件，上图的代码中我们会查询 user_id 为 xx 的用户的 tweets。

查找 SELECT

实际上我们在查询数据时不会把所有数据都返回，而是每次只返回一部分数据，并且按照时间顺序返回。

```
SELECT * FROM tweets_tweet where user_id = 1 ORDER BY created_at DESC LIMIT 10 ;
```

SQL

```
queryset=Tweet.objects.filter(user_id=user_id).order_by('-created_at')[:10]
```

Django ORM

LIMIT 表示每次查询返回10条数据，**OFFSET** 表示偏移量为0，即从第一条数据开始返回。

ORDER BY created_at 子句就是指定根据 created_at 字段排序，在排序方法上，有

DESC 降序排序和 **ASC 升序**排序。

通过上面的 SQL 语句便可以查询到小王最新的10条推文。



修改 UPDATE

看到小王学习打卡的推文，小张默默给小王点了一个赞。

此时就需要在数据库给小王被点赞的推文的 like_count 字段值 +1

```
UPDATE tweet SET likes_count = likes_count + 1 WHERE id = 1 ;
```

SQL

```
from django.db.models import F
Tweet.objects.filter(id=xx).update(like_count = F('like_count') + 1)
```

Django ORM

UPDATE 关键字表示这是一个更新操作，后面跟表名 tweet。**SET** 关键字用来设置字段的值，我们在 like_count 的基础上自增+1，**WHERE** 子句筛选哪些行需要被修改。通过上面的 SQL 语句，实现了，从 tweet 表中查找 id 值为1的行，并把 like_count 值自增+1。



删除 DELETE

看到小王日以继夜，孜孜不倦地学习九章地课程，不断地提升自己，小张在心里暗下决心，也要报名九章算法的课程，努力改变自己。小张打开了自己的推特，删除了自己昨天发布的游戏推文，要告别过去的自己。

```
DELETE FROM tweet WHERE id=xx
```

SQL

```
Tweet.objects.get(id=xx).delete()
```

Django ORM

DELETE FROM 关键字表示这是一个删除操作，其后跟表名表示要删除的表是 tweet 表，通过 **WHERE** 子句找到所有自己发的推文。

并默默打开了九章算法的官网。



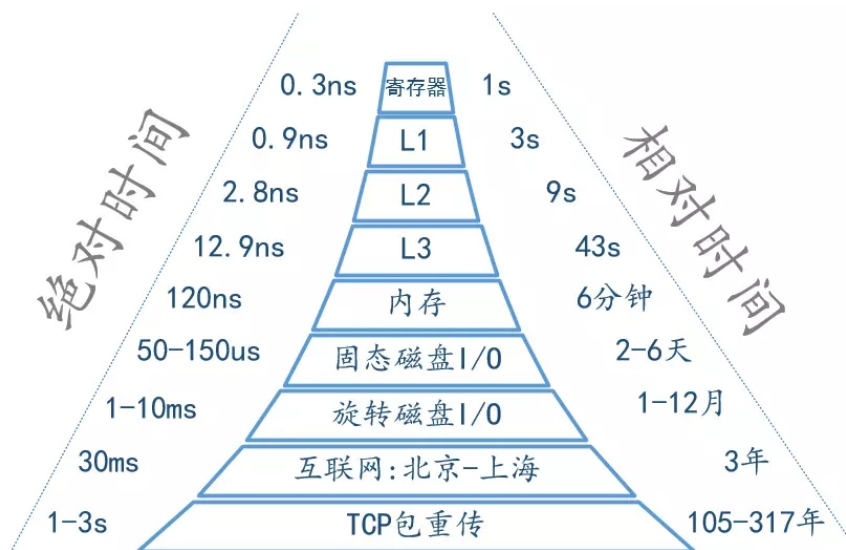
慢查询

以左边的推文 tweet 表为例，我们要查找表中数据的时候，就需要去扫描全表找到所有符合 created_at 在2021-01-01用户记录。

```
select * from tweet where created_at > '2021-01-01' and created_at < '2021-01-02';
```

tweets_tweet

- id: int
- content longtext
- created_at datetime(6)
- user_id: int
- comments_count bigint
- likes_count bigint



但是，MySQL的数据以页为基本单位存储在硬盘上 (页大小为16KB)。查询时，将页从硬盘读取到内存，再做查询。如果表中的数据记录很多，例如表大于16MB，就有1000个页需要被读取到内存中，但是硬盘的读取速度很慢，而页在硬盘中是不连续的，就会产生很多的随机IO，使得查询效率低下。

导致全表扫描速度耗时漫长的原因如下：

- 1 硬盘和内存之间的速度鸿沟
- 2 页的非顺序存储


但是硬盘与内存之间的速度差异客观存在，我们只能从“页”的角度想办法了

改页为顺序？

增大每页的容量？

接下来要介绍的索引就一种新思路——减少读取的页数了

tweets_tweet

 id: int
content longtext
◇ created_at datetime(6)
◇ user_id: int
comments_count bigint
likes_count bigint

索引 Index

我们可以给 created_at 字段创建一个索引，你可以理解为我们根据 created_at 进行了排序，但我们根据 created_at 进行查询时，就不需要再进行全表扫描了。

```
select * from tweet where created_at > '2021-01-01' and created_at < '2021-01-02';
```

联合索引 Compound index

有些时候，我们的查询条件不只一个字段，设想一个应用场景，查找某个用户发的最新的10条推文

```
SELECT * FROM tweet where user_id = xxx ORDER BY create_at DESC limit 10 ;
```

此时如果没有建立索引，就需要走全表扫描，再将所有符合`user_id = xxx`的记录，根据`create_at`字段降序排序，再取前十条，过于复杂和漫长。

如果只为`user_id`建立了索引，那么所有符合`user_id = xxx`的记录都将执行回表操作，再将结果根据`create_at`字段降序排序，取前十条。这种方式也有一个问题，就是回表操作过多，效率也不高。

此时，最好有一种索引，可以先根据`user_id`排序，在根据`create_at`排序，便可以大大降低回表的次数，提高效率，这种索引被称为联合索引。

联合索引 Compound index

```
class Tweet(models.Model):
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    photos = models.ManyToManyField(Photo, blank=True)
    ...
    likes_count = models.BigIntegerField(default=0, null=True)
    comments_count = models.BigIntegerField(default=0, null=True)

    class Meta:
        index_together = (('user', 'created_at'),)
```

```
CREATE INDEX tweet_user_id_created_at ON tweet (user_id, created_at);
```

explain

explain可以查看执行计划，下面对比有联合索引和没有联合索引时，执行计划的区别。

```
explain SELECT * FROM tweets_tweet where user_id = 1 order by created_at desc limit 10
```

没有联合索引

```
mysql> explain SELECT * FROM tweets_tweet where user_id = 1 order by created_at desc limit 10 \G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: tweets_tweet2
  partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
        key_len: NULL
         ref: NULL
         rows: 1
   filtered: 100.00
      Extra: Using where; Using filesort
1 row in set, 1 warning (0.00 sec)
```

有联合索引

```
mysql> explain SELECT * FROM tweets_tweet where user_id = 1 order by created_at desc limit 10 \G;
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: tweets_tweet
  partitions: NULL
         type: ref
possible_keys: tweets_tweet_user_id_created_at_d07f7061_idx
          key: tweets_tweet_user_id_created_at_d07f7061_idx
        key_len: 5
         ref: const
         rows: 1
   filtered: 100.00
      Extra: Backward index scan
1 row in set, 1 warning (0.00 sec)
```

索引失效

索引往往让我们的查找速度变的更快，但想让索引在查询中起作用，必须正确的创建和使用索引

让我们来看看一些导致索引失效的常见案例吧！

基于成本统计

mysql在执行查询之前，会生成一个查询计划。计算全表扫描的代价，计算使用不同索引执行查询的代价，对比各种执行方案的代价，找出成本最低的那一个。

因为索引的查询需要回表，所以如果回表次数过多，可能会拖慢我们的查询速度。根据查询计划，如果使用索引的查询回表次数过多，成本高于全表扫描，那么将不会使用索引查询，转而使用全表扫描。

索引失效

like通配符

当我们使用like做模糊匹配的时候，如果将%放在了左侧将导致索引失效

```
select * from userprofile where nickname = '%ck'
```

索引失效

```
select * from userprofile where nickname = 'j%ck'
```

索引可能命中

```
select * from userprofile where nickname = 'ja%'
```

索引可能命中

nickname是一个varchar 类型，当我们nickname字段建立二级索引的时候，就会创建一颗专属nickname的b+树，并按照字符串排序。通配符在最左侧将无法使用字符串排序，索引将失效。而另外两种情况，依据成本统计，有可能会走索引。

Mysql支持的非常多的字符集和排序规则，排序规则各不相同

tis620
ucs2
ujis
utf8
utf8mb4
utf16
utf16le
utf32

utf8mb4_0900_ai_ci
utf8mb4_0900_as_ci
utf8mb4_0900_as_cs
utf8mb4_bin
utf8mb4_croatian_ci
utf8mb4_cs_0900_ai_ci

索引失效

左前缀原理

联合索引是根据字段顺序进行排序的，当我们使用联合索引时，必须要根据联合索引的字段顺序设置where 查询子句

newsfeeds_newsfeed

- 🔑 id: int
- ◇ created_at: datetime(6)
- ◇ tweet_id: int
- ◇ user_id: int

如左图所示，为 (user_id, created_at) 创建一个联合索引 (Compound index)，下面两条语句中，第一条语句可以使用到联合索引，但第二条语句将使得联合索引失效

```
select * from newfeed where user_id = 1 and created_at > '2021-01-01' ;
```

```
select * from newfeed where created_at > '2021-01-01' ;
```

以下情况也会导致索引失效：

```
select * from newfeed where user_id < 10 and created_at > '2021-01-01' ;
```

在左侧的字段使用了范围索引的情况下，右侧的字段将无法使用索引

索引的代价

既然索引能提高查询效率，那么直接给所有列加上索引可好？

1. 索引文件本身消耗空间
2. 插入、更新、删除数据时，维护索引结构，添加负担
3. MySQL 自身运行时也要消耗资源维持索引

哪些列不要建索引?

1. 不太会查询到的列
2. 本身没有顺序，或者特别长的列。
3. 表记录比较少，就不用建索引了，直接全表扫描。
4. 重复率高，如「性别」。