# 一 索引

### 1.1 索引介绍

索引是一种有效组合数据的方式，通过索引对象，可以快速查询到数据库对象表中的特定记录，是提高性能的常用方式。

索引其实和书的目录类似，索引创建在表对象上，由表中的一个或者多个字段生成的键组成，键以B-树或者哈希表存储，所以也分为B树索引和哈希索引。InnoDB和MyISAM支持B-树索引，MEMORY支持哈希索引。

索引虽然提高了速度，但是也响应增加了存储空间，数据检索如果大于100ms就可以开始考虑优化了。主要优化where后面的语句。

下列字段经常作为索引：

经常被查询的字段，即在where中经常出现的字段

分组的字段，即在group by中出现的字段

存在依赖关系的子表和父表之间的联合查询，即主键和外键字段

设置唯一完整性约束的字段

以下字段不适合创建索引：

在查询中很少被使用的字段；

拥有许多重复值的字段。

### 1.2 索引使用

**创建表时创建普通索引：**

create table 表名 (

字段名 数据类型,

....

index [索引名](字段名 [长度] [asc|desc])

);

注意：因为不同引擎定义了表最大索引数和最大长度，所以可以设长度，mysql所支持的存储引擎对每个表至少支持16个索引，总索引长度至少为256个字节。

索引名最好以index\_开头。

**检测索引：**

explain select \* from users where name='lisi';

如果name被建立了索引，则查询速度变快。

**在已经存在的表上创建普通索引：**

create index 索引名 on 表名(字段名 [(长度)] [asc|desc]);

create index index\_name on t\_users(name);

**使用alter table创建普通索引：**

alter table 表名 add index 索引名(字段名 [(长度)] [asc|desc]);

**删除索引：**

drop index 索引名 on 表名;

### 1.3 唯一索引

唯一索引就是在创建索引时，限制索引的值必须是唯一的。在MySQL中，根据创建索引的方式，可分为自动索引和手动索引。

自动索引：在数据库表里设置完整约束时，该表会被系统自动创建索引；

手动索引：手动在表上创建索引。

**创建表时创建唯一索引：**

create table 表名 (

字段名 数据类型,

....

unique index [索引名](字段名 [(长度)] [asc|desc]);

);

**在表上创建唯一索引：**

create uinque index 索引名 on 表名 (字段名 [(长度)] [asc|desc]);

**alter table创建唯一索引：**

alter table 表名add unique index 索引名 (字段名 [(长度)] [asc|desc]);

### 1.4 全文索引

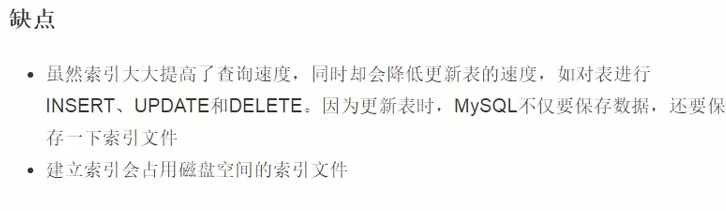
全文索引主要关联自数据类型为字符串的字段上，以便能够快速的查询数据量较大的字符串类型的字段，只有MyISAM引擎支持。默认情况下，全文索引的执行不区分大小写，如果全文索引所关联的字段为二进制数据类型，则区分大小写。

全文索引与唯一索引同样有三种创建方式，区别是使用关键字 fulltext代替unique。

### 1.5 多列索引

多列索引是指在创建索引时，关联了多个字段，但是只有查询条件中使用了所关联字段中的第一个字段，多列索引才会被使用。

创建方式同样有三种，只是字段处可以添加多个。



1 全文本搜索

通过正则、通配符可以实现复杂的搜索，但是随着搜索行数的增加，搜索将会变得耗时，且并不能智能的选择结果，比如：一个特殊词的搜索将会返回包含该词的所有行，而不区分包含单个匹配的行和包含多个匹配的行，类似的，一个特殊词的搜索将不会找出不包含该词但包含其他相关词的行。所有这些限制都可以通过全文检索来实现。

并非所有引擎支持全文本搜索，InnoDB不支持，MyISAM支持。

1.1 启用全文本搜索

为了进行全文本搜索，必须索引被搜索的列，而且要随着数据的改变不断重新索引。在堆表列进行适当设计后，MySQL会自动进行所有索引和重新设计。在索引之后，select可与Match()和Against()一起使用以实际执行搜索。

一般在创建表时启用全文本搜索：

create table products(

product\_id int not null auto\_increment,

product\_num char(10) not null,

product\_date datetime not null,

product\_text text null,

primary key(product\_id),

fulltext(product\_text)

)

engine=MyISAM;

fulltext索引单个列，如果需要也可以指定多个列。在定义之后，MySQL自动维护，增删改时，随之自动更新。

可以在创建时候指定fulltext，也可以稍后指定（此时必须所有已有数据立即索引）。

注意：更新索引需要花时间，如果正在导入数据到一个新表，此时不应该启用fulltext索引，应该先导入所有数据，然后再修改表，定义fulltext，这样有助于快速导入数据。

1.2 行全文本搜索

在索引后，使用Macth()和Against()执行全文本搜索，Match()指定被搜索的列，Against(0指定要使用的搜索表达式。

select product\_id from products

where Match(product\_text) Against('k');

# 二 视图

### 2.1 视图介绍

对于复杂的查询，在多次使用后，维护是一件非常麻烦的事情，可以使用视图查询，视图本质就是对查询的一个封装。

视图的特点：

视图的列可以来自不同的表，是表的抽象和在逻辑意义上建立的新关系；

视图是由数据库中的表产生的不存在的虚表；

视图的建立和删除不影响表本身；

对视图进行更新则会影响表；

当视图来自多个表时，不允许添加和删除数据。

### 2.2 视图创建和使用

create view 视图名 as 查询语句; //创建视图

select \* from 视图; //使用视图

注意：视图名一般以view\_或者v\_开头，视图名不能和表名、其他视图名重复

定义视图

create view stuscore as

select students.\*,scores.score from scores

inner join students on scores.stuid=students.id;

查询

select \* from stuscore;

### 2.3 查看视图

show table status [from 表名];

show create view 视图名; //用来查看视图创建语句

describe|desc 视图名; //用来查看视图设计信息

在系统数据库information\_schema中，也存储了视图信息，使用该表后，也可以查看视图：

select \* from views where table\_name='';

### 2.4 删除视图

drop view 视图名,视图名,....;

### 2.5 修改视图

create or replace view 视图名 as 查询语句;

或者：

alter view 视图名 as 查询语句;

### 2.6 利用视图操作数据

添加数据：insert into 视图名 (字段1,字段2...) values(值1,值2,...);

注意：视图来自多表不可添加删除数据。视图与字段已经对应，所以字段可以省略。

删除数据：delete from 视图名 where ...;

更新数据：update 视图名 set 字段1=\*\*,字段2=\*\*,...;

# 三 触发器

### 3.1 触发器介绍

触发器是mysql数据库对象之一，该对象与编程语言中的函数相似，需要声明，但是执行时靠事件触发。

业务场景：

添加一条学生记录，学生总数就要增加；

添加一条订单信息时，就要检查订单电话是否正确；

由此可见，触发器一般在执行delete、insert、update时候会自动触发。

### 3.2 创建触发器

create trigger 触发器名

after|before event

on 表名 for each row

trigger\_STMT;

触发器名一般以tri\_开头，且不能重复；

before|after是触发器执行时机；

event指触发条件：update delete insert；

trigger\_STMT指触发器激活后执行的语句。

案例：

create trigger tri\_ageup

after insert

on t\_father for each row

insert into t\_son values('zz',12);

注意：不能操作同一个表，因为会出现死循环。

### 3.3 创建包含多条执行语句的触发器

delimiter $$

create trigger 触发器名

after|before event

on 表名 for each row

begin

trigger\_STMT;

trigger\_STMT;

...;

end;

注意：delimiter $$是为了避免 触发器执行语句的 ; 作为了结束标志。

### 3.4 查看触发器

show trigger;

也可以在系统表中查询：

use infromation\_schema;

select \* from triggers;

### 3.5 删除触发器

drop trigger 触发器名;

# 四 存储过程

## 1 存储过程介绍

一个完整的数据库操作包含多条SQL语句，在执行过程中需要根据前面的SQL结果有选择的执行后面的SQL，此时可以使用存储过程。

存储过程可以简单的理解为一条或者多条SQL的集合，与函数的区别是：函数有返回值，而存储过程没有。

存储过程的优点：

1 提升SQL的重用性、可移植性

2 能够实现较快的执行速度，减少网络流量

3 可以作为一种安全机制来利用

## 2 创建存储名为test的过程

create procedure test()

begin

select Avg(product\_price) as priceavg from products;

end;

注意：默认mysql分隔符为 ; ，如果使用的是mysql命令行，要解析存储过程中自身的 ; ，会出现语法错误，解决办法：使用delimiter关键字。

delimiter //

create procedure test()

begin

select Avg(product\_price) as priceavg from products;

end //

这里使用 //代替 ;，除了 / 外，都可以作为分隔符。

## 3 存储过程的使用与删除

call test(); //使用

drop procedure test; //删除：如果不存在该存储过程，会出现错误。

## 4 带参数的存储过程

create procedure productpriceing(

out p1 DECIMAL(8,2),

out ph DECIMAL(8,2),

out pa DECIMAL(8,2)

)

begin

select Min(product\_price) into p1 from products;

select Max(product\_price) into ph from products;

select Avg(product\_price) into pa from products;

end;

解析：该存储过程接收3个参数，p1存储最低价格，ph存储最高价格，pa存储平均价格，每个参数必须具有指定的类型，这里是十进制。关键字out指出对应参数用来从存储过程中传出一个值给调用者。

in：传递给存储过程

out：传出给调用者

inout：传入和传出

使用带参数存储过程：

call productpricing(

@pricelow,

@pricehigh,

@priceavg

);

mysql的变量使用@，该语句并不现实任何数据，只返回以后可以显示的变量，如果药检所产品的平均价格，如下进行：

select @priceavg;

# 五 事务

## 1 事务简介

当一个业务逻辑需要多个sql完成时，如果其中某条sql语句出错，则希望整个操作都退回，使用事务可以完成退回的功能，保证业务逻辑的正确性。比如银行的转账，任意一个步骤出错，就应该全部停止回滚。

事务四大特性(简称ACID)：

1-原子性(Atomicity)： 事务的全部操要么全部完成，要么均不执行；

2-一致性(Consistency)： 并行执行的多个事务，执行结果必须与串行执行结果一致；

3-隔离性(Isolation)： 事务的执行不受其他事务的干扰，但中间结果对其他事务透明的

4-持久性(Durability)： 已提交事务，系统必须保证该事务对数据库的改变不被丢失。

注意： 表必须是innodb或bdb类型，才可以对此表使用事务，一般默认是innodb。

inndb原理：对当前操作的行加锁；适合插入增删改（写入速度快）；

其他引擎常见的是对整张表加锁，查询速度很快。

## 2事务的使用

在mysql中事务都是自动提交的，通过语句查看是否自动提交：

show variables like ‘%commit%’;

set autocommit=off; //该语句可以关闭自动提交

事务常用命令：begin commit rollback

事务示例：

步骤1：打开两个终端，连接mysql，使用同一个数据库，操作同一张表

终端1：

select \* from users;

------------------------

终端2：

begin;

insert into users(name) values('张飞');

步骤2：

终端1：

select \* from users;

步骤3:

终端2：

commit;

------------------------

终端1：

select \* from users;

## 3 隔离级别

### 3.0 事务的安全隐患

SQL标准定义了4种隔离级别，指定了事务中哪些数据改变其他事务课件，哪些数据改变其他事务不课件。低级别的隔离级别可以支持更高的并处理，同时占用的系统资源更少。

查看隔离级别： select @@tx\_isolation

设置隔离级别： set session transaction isolation level 隔离级别名；

（session也可以改为global，即全局的隔离级别）

4个级别： read uncommitted 未提交读

read commited 提交读 （oracle默认隔离级别）

repeatable read 可重复读 （mysql默认隔离级别）

serializable 可串行化

### 3.1 读未提交--脏读

该事物隔离级别低，且响应提升性能空间少，所以用的较少。

脏读： 一个事务读到了另外一个事务还未提交的数据。

某个事务已经更新了一个数据，但是另一个事务在此时读取了一份数据，前一个事务回滚，后一个事务就会读取到不存在的数据；

### 3.2 读已提交--不可重复读

提交读是大多数数据库默认的隔离级别，但是不是mysql的默认级别。该级别是人们比较常能理解的：一个事务在开始到提交前所做的任何改变都是不可见的，只能等待提交后才能看到。

在MySQL中，开启A事务和B事务，B事务提交后，A事务能够读取到更新的数据。

虽然这个隔离级别屏蔽了脏读，但是这样会造成不可重复读：一个事务读取到了另外一个事务提交的数据，造成前后两次查询结果不一致。

### 3.3 可重复读--幻读

MySQL默认隔离级别：确保同一事务的不同实例在并发读取数据时，会看到相同的数据行。

但是会出现幻读：一个事务读到了另一个事务insert的数据，造成前后查询结果不一致。

第1个事务实例对表中的所有数据进行了修改；

第2个事务实例向表中插入一行；

以后会发生操作第1个事务的用户发现表中存在没有被修改的行。

### 3.4 可串行化

在每个读的数据上加共享锁，强制事务排序，避免幻读，但是会导致大量的超时、锁竞争现象，一般不推荐使用。

### 3.5 写隐患--丢失更新

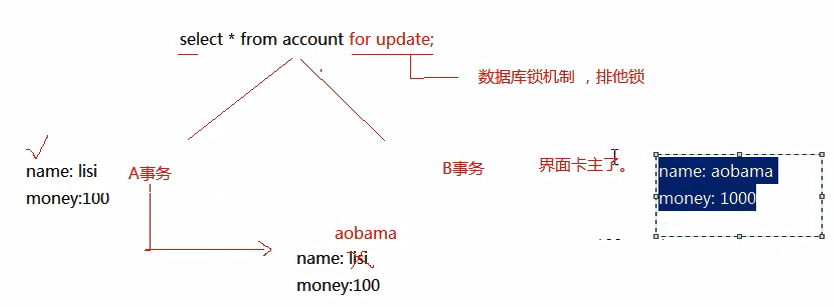
第二个事务的update覆盖了第一个事务的update

## 4 锁

锁用来解决丢失更新的问题。

### 4.1悲观锁

查询的时，认为一定会出现丢失更新，那么 查询语句后 加入 for update（排他锁）



### 4.2 乐观锁

乐观锁：要求程序员自己控制，额外添加一个字段，比如版本 version:0

第二个事务如果发现版本不一样，则重新查。

