索引（index）

数据库索引：索引是一种特殊的数据结构，可以用来快速查询数据库表中的特定记录。索引是提高数据库性能的重要方式，所有字段都可以添加索引。

索引包括：普通索引、唯一性索引、全文索引、单列索引、多列索引、空间索引。

使用索引可以提升检索数据的速度，但是创建和维护索引需要消耗时间，索引需要占用物理空间。

普通索引：不需要任何限制条件的索引，可以在任意的数据类型上创建。

唯一索引：索引的值必须唯一，比如主键索引。

全文索引：只能创建在char、varchar、text类型的字段上、查询数据量较大的字符串类型字段时，使用全文索引可以提高速度。

InnoDB存储引擎不支持全文索引。

单列索引：只对应一个字段的索引。

多列索引：在一张表的多个字段上创建一个索引，查询时使用第一个字段，即可触发该索引。

空间索引：只能建立在空间数据类型上（GIS），InnoDB存储引擎不支持空间索引。

- 主键自带索引。
- 可以给其他字段添加索引，但是不能盲目给每个字段都添加，反而会适得其反，应该将索引添加在where后面的字段上，select后面的字段不要加索引。

索引的设计原则：

- 出现在where语句中的列，select语句中的不加。
- 索引的值尽量唯一，效率更高。
- 不要添加过多的索引，维护成本很高。

添加索引

```
alter table user add index in_name(name);
create index in_age on user(age);
```

删除索引

```
alter table user drop index in_name;
drop index in_age on user;
```

事务（transaction）

事务：将多条SQL作为一个整体来执行，要么全部执行成功，要么一条都不执行。

例子：张三和李四各有1000块钱，张三借给李四500块钱，

name money

张三 1000

李四 1000

王五 1000

张三借钱给李四的操作需要执行两次SQL：

1.update user set money = 500 where name="张三";

2.update user set money = 1500 where name = "李四";

张三借钱给王五

1.update user set money = 0 where name="张三";

2.update user set money = 1500 where name = "李四";

事务的四个特性：

- 原子性：多条SQL语句是一个整体，不可在分割。
- 一致性：SQL语句执行前后，数据库数据的值保持一致。
- 隔离性：一个事务的执行不能被其他事务干扰。
- 持久性：一旦事务提交，数据库中的数据的改变是永久性的。

对数据库的每一次操作（除了查询）都必须提交事务才能完成持久化工作，MySQL是默认开启事务自动提交的，我们只需要执行SQL，事务会自动提交。

查看数据库是否开启自动提交

```
show variables like "autocommit";
```

修改数据库事务的自动提交

```
set autocommit = 0;//关闭自动提交
set autocommit = 1;//开启自动提交
```

事务提交

```
update user set money = 500 where name = "张三";
update user set money = 1500 where name = "李四";
commit;
```

事务回滚

```
update user set money = 500 where name = "张三";
update user set money = 1500 where name = "李四";
rollback;
```

视图（view）

数据库中虚拟的一张表，允许不同用户或者应用程序以不同的方式查看同一张表中的数据。

有的人可以看到工资，有的人不能看到工资。

- 创建两张表，一张有工资字段，另外一张没有工资字段，除了工资字段之外其他字段完全一致。有工资的表给财务人员看，没有工资的表给普通职员看。
- 为不同的用户创建不同的视图，视图是一张虚拟的表，它里面的字段和数据来自一张真实存在的表，在创建视图时可以选择展示哪些字段和对应的值。

创建视图

```
create view view_common as select name,age from user;
```

使用视图

```
select * from view_all;
```

删除视图

```
drop view view_common;
```

触发器(trigger)

触发器定义了一系列的操作，可以对指定表进行插入、更新或者删除操作的同时自动执行这些操作。

触发器的优点：

- 开发更快，因为触发器存储在数据库中，所以不必编写每个触发器在应用程序中执行的操作。
- 更容易维护，定义触发器之后，访问目标表会自动调用触发器。
- 业务全局实现，如果修改业务，只需要修改触发器即可，不需要修改业务代码。

触发器的分类：

- 前触发器：在更新或插入操作之前运行
- 后触发器：在更新、插入、删除之后运行
- Before delete触发器：在删除之前执行
- Insted of 触发器：对复杂视图执行插入，更新和删除时运行

创建触发器

```
create trigger t_afterinsert_on_tab1
  after insert on tab1
  for each row
  begin
    insert into tab2(tab2_id) values (new.tab1_id);
  end;
```

```
create trigger t_afterdelete_on_tab1
  after delete on tab1
  for each row
  begin
    delete from tab2 where tab2_id = old.tab1_id;
  end;
```

删除触发器

```
drop trigger t_afterinsert_on_tab1;
```

存储过程(procedure)

存储过程是一组为了完成特定功能的sql语句的机会，经过编译存储在数据库中，用户通过指定存储过程的名字并给出参数来执行。一次编写多次调用，避免开发人员重复编写相同的SQL，存储过程是在数据库存储和执行的，可以减少客户端和服务器之间的数据传输，提高效率。

优点：

- 模块化的程序设计，只需要创建一次存储过程，以后就可以在程序中调用该存储过程，多次调用。
- 执行速度更快，如果某操作需要执行大量的SQL语句，存储过程比普通的SQL执行速度更快。
- 更好的安全机制，对于没有权限执行存储过程的用户，可以通过授权的方式来执行存储过程。

创建存储过程的基本语法：

create procedure sp_name ([proc_paramter[...]])

routine_body

sp_name：存储过程的名字

proc_paramter：参数列表

三部分组成：输入输出类型，参数名称，参数类型

in表示入参，out表示出参，参数类型为MySQL数据库的任意数据类型

routine_body：SQL语句

begin/end 来标识SQL语句的开始和结束

入参的存储过程

```
use demo;
create procedure add_name(in target int)
  begin
    declare name varchar(20);
    if target = 1 then
      set name = "MySQL";
      else
      set name = "Java";
    end if;
    insert into user(name) values (name);
  end;
```

调用存储过程

```
call add_name(11);
```

删除存储过程

```
drop procedure add_name;
```

出参存储过程

```
create procedure count_of_user(out count_num int)
  begin
    select count(*) into count_num from user;
  end;
```

调用

```
call count_of_user(@cout_num);
select @cout_num;
```

流程控制语句

if

```
create procedure example_if(in x int)
  begin
    if x = 1 then
      select name from user;
      elseif x = 2 then
      select age from user;
    end if;
  end;
```

case

```
create procedure example_case(in x int)
  begin
    case x
      when 1 then select name from user;
      when 2 then select age from user;
      else select money from user;
      end case;
  end;
```

while

```
create procedure example_while(out sum int)
  begin
    declare i int default 1;
    declare s int default 0;
    while i<=100 do
      set s=s+i;
      set i=i+1;
    end while;
    set sum = s;
  end;
```