MySQL是关系型数据库

运行mysql服务：net start mysql80 （管理员身份运行cmd）

关闭mysql服务：net stop mysql80

连接MySQL：

1. 客户端 Client连接
2. 执行命令：mysql -h 127.0.0.1 -P 3306 -u root -p

mysql -u root -p

本机连接密码：fantastic0918

CentOS7的MySQL连接密码：fantastic0918

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

1. 数据库（DB）：如database1，dat2；数据库管理系统（DBMS）：如MySQL，Oracle
2. 注释：（1）单行注释：--注释内容 或 #注释内容 （MySQL特有）

（2）多行注释：/\* 注释内容 \*/

1. SQL分类
2. DDL 定义 定义数据库对象 Definition
3. DML 操作 增删改 Manipulation
4. DQL 查询 查询记录
5. DCL 控制 创建数据库用户、控制数据库的访问权限
6. 常用类型
   1. tinyint-1字节；smallint-2字节；mediumint-3字节；int-4字节
   2. float-4字节；double-8字节 数值类型后可跟unsigned
   3. char(3)-定长3字节；varchar(10)-变长最大10字节；char的性能更高
   4. data-3字节，年月日；time-3字节，时分秒；datetime-8字节，混合
7. **DDL：定义**  //在表中，字段表示一列

（1）show databases ;

（2）select database() ; //查询当前处于哪个数据库

（3）create database （[if not exists]） shujuku1 （[ default charset utf8mb4]）; //utf8mb4:每个字符占四个字节

#### （4）drop database [ if exists ] 数据库名 ; //删除数据库

（5）use 数据库名 ;

（6）show tables;

（7）desc 表名 ; //查询表结构

#### （8）show create table 表名 ; //查询建表语句

（9）create table 表名（ //建表语句

字段1 字段1类型 [comment ‘字段1注释’],

...,

字段n 字段n类型 [comment ‘字段n注释’]

）[comment ‘注释’];

1. alter table emp add nickname varchar(20) comment '昵称';

//添加列（字段）

（11）alter table emp modify nickname varchar(30) comment ‘昵称’； //修改列的类型

（12）alter table emp change nickname username varchar(30) commment ' 昵 称 '; //修改列的名字和类型

（13）alter table emp drop username; //删除列

（14）alter table emp rename to employee; //修改表名

（15）drop table if exists emp; //删除表

（16）truncate table emp; //删除并重新创建该表

1. **DML：操作**
2. insert （插入数据）

insert into emp(id,workno,name,gender,age,idcard,workaddress,entrydate) values (17,'00017','汤姆','男',22,'454622222222180855','杭州','2001-02-03');

批量添加：

insert into emp (字段名1, 字段名2, ...) values (值1, 值2, ...), (值1, 值2, ...), (值1, 值2, ...) ;

INSERT INTO 表名 VALUES (值1, 值2, ...), (值1, 值2, ...), (值1, 值2, ...) ;

1. update （修改已经插入的数据） //没有条件表示修改全表

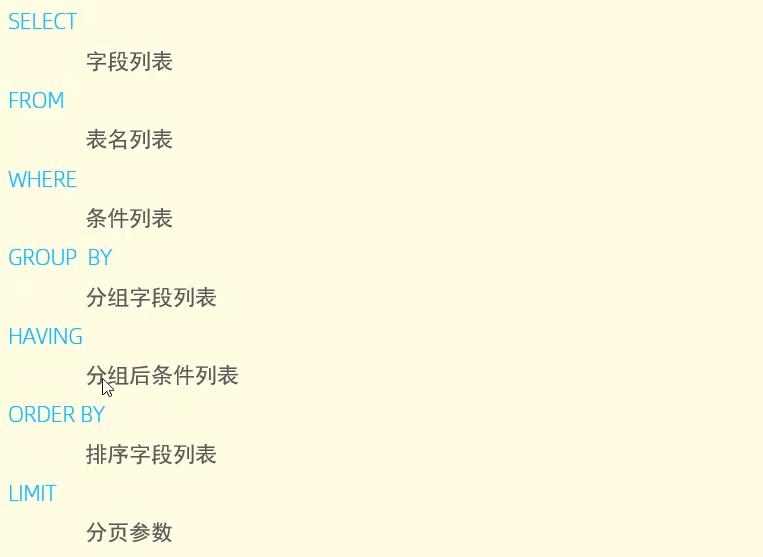
update emp set name = ‘小白’where id = 1 ;

1. delete //删除行，没有条件表示删除本表的所有数据

delete from employee where gender = '女';

1. **DQL：查询**

**DQL语法：**



执行顺序：from->where->group by->select->order by->limit

select distinct 字段1 as’别名’, 字段2, ... from emp ;

//在select后加distinct可去除查询到的重复记录

where-条件查询（WHERE）

比较符：>,>=,<,<=,=,<>或!=,between..and..包含边界,in(1,4)符合1或4，like ‘\_%’\_占一位，%占多位,is null；逻辑：and-&&，or-||，not-！

group by-分组查询

分组查询的字段一般为聚合函数和分组字段，查询其他字段无任何意义。

按多段分组记录条数就是分组内所有种类的笛卡尔乘积

order by-排序查询

asc，desc

limit-分页查询

limit 0,10

7.1聚合函数（count、max、min、avg、sum）--对列使用

count:统计不为null的数量，不是相加

select count(\*) from emp; -- 统计的是总记录数

max:最大值

select max(age) from emp;

min:最小值

select min(age) from emp;

avg:平均值

select avg(age) from emp;

sum:求和

select sum(age) from emp where workaddress = '西安';

7.2分组查询

SELECT 字段列表 FROM 表 名 [ WHERE 条件 ] GROUP BY 分组字段名 [ HAVING 分组后过滤条件 ];

select workaddress, gender, count(\*) '数量' from emp group by gender , workaddress;

where与having区别：（1）执行时机不同：where是分组之前进行过滤，不满足where条件，不参与分组；而having是分组 之后对结果进行过滤。（2）判断条件不同：where不能对聚合函数进行判断，而having可以。

5.3排序查询

SELECT 字段列表 FROM 表名 ORDER BY 字段1 排序方式1 , 字段2 排序方式2 ;

select \* from emp order by age asc , entrydate desc;

5.4分页查询

SELECT 字段列表 FROM 表名 LIMIT 起始索引, 查询记录数 ;

select \* from emp limit 0,10;

起始索引从0开始，起始索引 = （查询页码 - 1）\* 每页显示记录数。

1. **DCL**

7.1管理用户

1. 查询有多少用户

select \* from mysql.user;

1. 创建用户

create user '用户名'@'主机名' identified by '密码';

1. 修改用户密码

alter user '用户名'@'主机名' identified with mysql\_native\_password by '新密码' ;

1. 删除用户

drop user '用户名'@'主机名' ;

### 7.2权限控制

MySQL中定义了很多种权限，但是常用的就以下几种：

|  |  |
| --- | --- |
| **权限** | **说明** |
| ALL, ALL PRIVILEGES | 所有权限 |
| SELECT | 查询数据 |
| INSERT | 插入数据 |
| UPDATE | 修改数据 |
| DELETE | 删除数据 |
| ALTER | 修改表 |
| DROP | 删除数据库/表/视图 |
| CREATE | 创建数据库/表 |

1. 查询权限

show grants for '用户名'@'主机名' ;

1. 授予权限

grant 权限列表 ON 数据库名.表名 TO '用户名'@'主机名';

1. 撤销权限

revoke 权限列表 ON 数据库名.表名 FROM '用户名'@'主机名';

1. 字符串函数

MySQL中内置了很多字符串函数，常用的几个如下：

|  |  |
| --- | --- |
| **函数** | **功能** |
| concat(S1,S2,...Sn) | 字符串拼接，将S1，S2，... Sn拼接成一个字符串 |
| lower(Str) | 将字符串str全部转为小写 |
| upper(str) | 将字符串str全部转为大写 |
| lpad(str,n,pad) | 左填充，用字符串pad对str的左边进行填充，使str达到n个字符串长度 |
| rpad(str,n,pad) | 右填充，用字符串pad对str的右边进行填充，使str达到n个字符串长度 |
| trim(str) | 去掉字符串头部和尾部的空格 |
| substring(str,start,len) | 返回从字符串str从start位置起的len个长度的字符串 |

substring：起始下标从1开始

例：

由于业务需求变更，企业员工的工号，统一为5位数，目前不足5位数的全部在前面补0。比如： 1号员工的工号应该为00001。

update emp set workno = lpad(workno, 5, '0');

1. 数值函数

常见的数值函数如下：

|  |  |
| --- | --- |
| **函数** | **功能** |
| cell(x) | 向上取整 |
| floor(x) | 向下取整 |
| mod(x,y) | 返回x/y的模 |
| rand() | 返回0~1内的随机数 |
| round(x,y) | 求参数x的四舍五入的值，保留y位小数 |

案例：

通过数据库的函数，生成一个六位数的随机验证码。

思路： 获取随机数可以通过rand()函数，但是获取出来的随机数是在0-1之间的，所以可以在其基础上乘以1000000，然后舍弃小数部分，如果长度不足6位，补0

1 select lpad(round(rand()\*1000000 , 0), 6, '0');

1. 日期函数

常见的日期函数如下：

|  |  |
| --- | --- |
| **函数** | **功能** |
| curdate() | 返回当前日期 |
| curtime() | 返回当前时间 |
| now() | 返回当前日期和时间 |
| year(date) | 获取指定date的年份,当前年:year(now()) |
| month(date) | 获取指定date的月份 |
| day(date) | 获取指定date的日期 |
| date\_add(date, interval expr year/month/day) | 返回一个日期/时间值加上一个时间间隔expr后的时间值 |
| datediff(date1,date2) | 返回结束时间date1 和 开始时间date2之间的天数,返回date1-date2 |

案例：

查询所有员工的入职天数，并根据入职天数倒序排序。

思路： 入职天数，就是通过当前日期 - 入职日期，所以需要使用datediff函数来完成。

1 select name, datediff(curdate(), entrydate) as 'entrydays' from emp order by

entrydays desc;

1. 流程函数

流程函数也是很常用的一类函数，可以在SQL语句中实现条件筛选，从而提高语句的效率。

|  |  |
| --- | --- |
| **函数** | **功能** |
| if(value , t , f) | 如果value为true，则返回t，否则返回f |
| ifnull(value1 , value2) | 如果value1不为空，返回value1，否则返回value2 |
| case when[ val1 ] then [res1] ...  else [ default ] end | 如果val1为true，返回res1，... 否则返回default默认值 |
| case [ expr ] when [ val1 ] then  [res1] ... else [ default ] end | 如果expr的值等于val1，返回  res1，... 否则返回default默认值 |

1 select

2

id,

3

name,

1. (case when math >= 85 then '优秀' when math >=60 then '及格' else '不及格' end ) '数学',
2. (case when english >= 85 then '优秀' when english >=60 then '及格' else '不及格' end ) '英语',
3. (case when chinese >= 85 then '优秀' when chinese >=60 then '及格' else '不及格'

end ) '语文'

7 from score;

1. 约束-作用于字段上-限制作用

目的：保证数据库中数据的正确、有效性和完整性。

|  |  |  |
| --- | --- | --- |
| **约束** | **描述** | **关键字** |
| 非空约束 | 限制该字段的数据不能为null | NOT NULL |
| 唯一约束 | 保证该字段的所有数据都是唯一、不重复的 | UNIQUE |
| 主键约束 | 主键是一行数据的唯一标识，要求非空且唯一 | primary key |
| 默认约束 | 保存数据时，如果未指定该字段的值，则采用默认值 | default |
| 检查约束(8.0.16版本之后) | 保证字段值满足某一个条件 | check |
| 外键约束 | 用来让两张表的数据之间建立连接，保证数据的一致性和完整性 | foreign key |

例：

CREATE TABLE tb\_user(

id int AUTO\_INCREMENT PRIMARY KEY COMMENT 'ID唯一标识',

name varchar(10) NOT NULL UNIQUE COMMENT '姓名' ,

age int check (age > 0 && age <= 120) COMMENT '年龄' ,

status char(1) default '1' COMMENT '状态',

gender char(1) COMMENT '性别'

);

为已有表设置主键：alter table emp add primary key(id);

注意：约束是作用于表中字段上的，可以在创建表/修改表的时候添加约束。

1. 语法
2. 添加外键

1 CREATE TABLE 表名(

2

字段名 数据类型,

3

...

4

[CONSTRAINT] [外键名称] FOREIGN KEY (外键字段名) REFERENCES 主表 (主表列名)

5 );

1 ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY (外键字段名) REFERENCES 主表 (主表列名) ;

案例:

为emp表的dept\_id字段添加外键约束,关联dept表的主键id。

1 alter table emp add constraint fk\_emp\_dept\_id foreign key (dept\_id) references

dept(id);

1. 删除外键

1 alter table emp drop foreign key fk\_emp\_dept\_id;

1. 删除/更新行为

添加了外键之后，再删除父表数据时产生的约束行为，我们就称为删除/更新行为。具体的删除/更新行 为有以下几种:

|  |  |
| --- | --- |
| **行为** | **说明** |
| no action | 当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有则不 允许删除/更新。 (与 RESTRICT 一致) 默认行为 |
| restrict | 当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有则不 允许删除/更新。 (与 NO ACTION 一致) 默认行为 |
| cascade | 当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有，则 也删除/更新外键在子表中的整条记录。 |
| set null | 当在父表中删除对应记录时，首先检查该记录是否有对应外键，如果有则设置子表 中该外键值为null（这就要求该外键允许取null）。 |
| set default | 父表有变更时，子表将外键列设置成一个默认的值 (Innodb不支持) |

具体语法为:

1 alter table emp add constraint fk\_emp\_dept\_id foreign key (dept\_id) references

dept(id) on update set null on delete set null ;

1. 多表查询

### 15.1一对多

实现: 在多的一方建立外键，指向一的一方的主键

### 15.2多对多

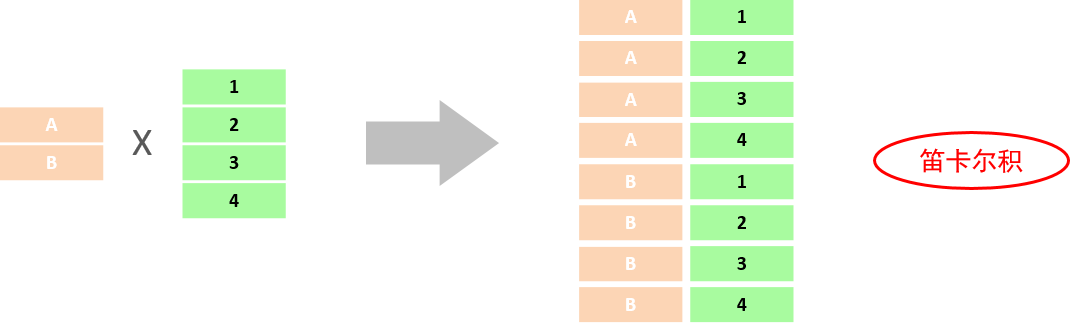
实现: 建立第三张中间表，中间表至少包含两个外键，分别关联两方主键

### 15.3一对一

实现: 在任意一方加入外键，关联另外一方的主键，并且设置外键为唯一的(UNIQUE)

1. 查询多表时会出现笛卡尔积:

笛卡尔乘积是指在数学中，两个集合A集合 和 B集合的所有组合情况。



在多表查询中，需要消除无效的笛卡尔积的行

1. 内连接

内连接查询的是两张表交集部分的数据。

1. 隐式内连接

1 SELECT 字段列表 FROM 表1 , 表2 WHERE 条件 ... ;

1. 显式内连接

1 SELECT 字段列表 FROM 表 1 [ INNER ] JOIN 表 2 ON 连 接 条 件 ... ;

1. 一旦为表起了别名，就不能再使用表名来指定对应的字段了，此时只能够使用别名来指定字段。

## 外连接

外连接分为两种，分别是：左外连接 和 右外连接。

1. 左外连接（查询左表1的所有数据，当然也包含表1和表2交集部分的数据。）

1 SELECT 字段列表 FROM 表 1 LEFT [ OUTER ] JOIN 表 2 ON 条 件 ... ;

右外连接（查询右表2的所有数据，当然也包含表1和表2交集部分的数据。）

1 SELECT 字段列表 FROM 表 1 RIGHT [ OUTER ] JOIN 表 2 ON 条 件 ... ;

1. 自连接

自连接查询，顾名思义，就是自己连接自己，也就是把一张表连接查询多次。我们先来学习一下自连接 的查询语法：

1 SELECT 字段列表 FROM 表A 别名A JOIN 表A 别名B ON 条 件 ... ;

而对于自连接查询，可以是内连接查询，也可以是外连接查询。

1. 在自连接查询中，必须要为表起别名，要不然我们不清楚所指定的条件、返回的字段，到底 是哪一张表的字段。
2. 联合查询

对于union查询，就是把多次查询的结果合并起来，形成一个新的查询结果集。

1. SELECT 字段列表 FROM 表 A ...
2. UNION [ ALL ]
3. SELECT 字段列表 FROM 表B ;

对于联合查询的多张表的列数必须保持一致，字段类型也需要保持一致。

union all 会将全部的数据直接合并在一起，union 会对合并之后的数据去重。

1. 子查询

SQL语句中嵌套SELECT语句，称为嵌套查询，又称子查询。

1 SELECT \* FROM t1 WHERE column1 = ( SELECT column1 FROM t2 );

子查询外部的语句可以是INSERT / UPDATE / DELETE / SELECT 的任何一个。但是括号里面只能是select语句

1. 标量子查询

子查询返回的结果是单个值（数字、字符串、日期等），最简单的形式，这种子查询称为标量子查询。 常用的操作符：= <> > >= < <=

select \* from emp where dept\_id = (select id from dept where name = '销售部');

1. 列子查询

子查询返回的结果是一列（可以是多行），这种子查询称为列子查询。常用的操作符：IN 、NOT IN 、 ANY 、SOME 、 ALL

|  |  |
| --- | --- |
| **操作符** | **描述** |
| IN | 在指定的集合范围之内，多选一 |
| NOT IN | 不在指定的集合范围之内 |
| ANY | 子查询返回列表中，有任意一个满足即可 |
| SOME | 与ANY等同，使用SOME的地方都可以使用ANY |
| ALL | 子查询返回列表的所有值都必须满足 |

查询 "销售部" 和 "市场部" 的所有员工信息

1 select \* from emp where dept\_id in (select id from dept where name = '销售部' or name = '市场部');

查询比 财务部 所有人工资都高的员工信息

1 select \* from emp where salary > all ( select salary from emp where dept\_id =

(select id from dept where name = '财务部') );

查询比研发部其中任意一人工资高的员工信息分解

1 select \* from emp where salary > any ( select salary from emp where dept\_id =

(select id from dept where name = '研发部') );

1. 行子查询

子查询返回的结果是一行（可以是多列），这种子查询称为行子查询。常用的操作符：= 、<> 、IN 、NOT IN

查询与 "张无忌" 的薪资且直属领导相同的员工信息 ;

1 select \* from emp where (salary,managerid) = (select salary, managerid from emp

where name = '张无忌');

1. 表子查询

子查询返回的结果是多行多列，这种子查询称为表子查询。常用的操作符：IN

查询与 "鹿杖客" , "宋远桥" 的职位和薪资相同的员工信息

1 select \* from emp where (job,salary) in ( select job, salary from emp where name =

'鹿杖客' or name = '宋远桥' );

查询入职日期是 "2006-01-01" 之后的员工信息 , 及其部门信息

1 select e.\*, d.\* from (select \* from emp where entrydate > '2006-01-01') e left

join dept d on e.dept\_id = d.id ;

1. 多表查询案例

数据环境准备:

1 create table salgrade(

2

grade int,

3

losal int,

4

hisal int

5 ) comment '薪资等级表'; 6

1. insert into salgrade values (1,0,3000);
2. insert into salgrade values (2,3001,5000);
3. insert into salgrade values (3,5001,8000);
4. insert into salgrade values (4,8001,10000);
5. insert into salgrade values (5,10001,15000);
6. insert into salgrade values (6,15001,20000);
7. insert into salgrade values (7,20001,25000);

14 insert into salgrade values (8,25001,30000);

在这个案例中，我们主要运用上面所讲解的多表查询的语法，完成以下的12个需求即可，而这里主要涉 及到的表就三张：emp员工表、dept部门表、salgrade薪资等级表 。

1. 查询员工的姓名、年龄、职位、部门信息 （隐式内连接） 表 : emp , dept

连接条件: emp.dept\_id = dept.id

1 select e.name , e.age , e.job , d.name from emp e , dept d where e.dept\_id = d.id;

1. 查询年龄小于30岁的员工的姓名、年龄、职位、部门信息（显式内连接） 表 : emp , dept

连接条件: emp.dept\_id = dept.id

1 select e.name , e.age , e.job , d.name from emp e inner join dept d on e.dept\_id =

d.id where e.age < 30;

1. 查询拥有员工的部门ID、部门名称表 : emp , dept

连接条件: emp.dept\_id = dept.id

1 select distinct d.id , d.name from emp e , dept d where e.dept\_id = d.id;

1. 查询所有年龄大于40岁的员工, 及其归属的部门名称; 如果员工没有分配部门, 也需要展示出来(外连接)

表 : emp , dept

连接条件: emp.dept\_id = dept.id

1 select e.\*, d.name from emp e left join dept d on e.dept\_id = d.id where e.age >

40 ;

1. 查询所有员工的工资等级表 : emp , salgrade

连接条件 : emp.salary >= salgrade.losal and emp.salary <= salgrade.hisal

1. -- 方式一
2. select e.\* , s.grade , s.losal, s.hisal from emp e , salgrade s where e.salary >= s.losal and e.salary <= s.hisal;
3. -- 方式二
4. select e.\* , s.grade , s.losal, s.hisal from emp e , salgrade s where e.salary between s.losal and s.hisal;
5. 查询 "研发部" 所有员工的信息及 工资等级表 : emp , salgrade , dept

连接条件 : emp.salary between salgrade.losal and salgrade.hisal , emp.dept\_id = dept.id

查询条件 : dept.name = '研发部'

1 select e.\* , s.grade from emp e , dept d , salgrade s where e.dept\_id = d.id and (

e.salary between s.losal and s.hisal ) and d.name = '研发部';

1. 查询 "研发部" 员工的平均工资表 : emp , dept

连接条件 : emp.dept\_id = dept.id

1 select avg(e.salary) from emp e, dept d where e.dept\_id = d.id and d.name = '研发

部';

1. 查询工资比 "灭绝" 高的员工信息。

①. 查询 "灭绝" 的薪资

1 select salary from emp where name = '灭绝';

②. 查询比她工资高的员工数据

1 select \* from emp where salary > ( select salary from emp where name = '灭绝' );

1. 查询比平均薪资高的员工信息

①. 查询员工的平均薪资

1 select avg(salary) from emp;

②. 查询比平均薪资高的员工信息

1 select \* from emp where salary > ( select avg(salary) from emp );

1. 查询低于本部门平均工资的员工信息

①. 查询指定部门平均薪资

1. select avg(e1.salary) from emp e1 where e1.dept\_id = 1;
2. select avg(e1.salary) from emp e1 where e1.dept\_id = 2;

②. 查询低于本部门平均工资的员工信息

1 select \* from emp e2 where e2.salary < ( select avg(e1.salary) from emp e1 where

e1.dept\_id = e2.dept\_id );

1. 查询所有的部门信息, 并统计部门的员工人数

1 select d.id, d.name , ( select count(\*) from emp e where e.dept\_id = d.id ) '人数'

from dept d;

1. 查询所有学生的选课情况, 展示出学生名称, 学号, 课程名称表 : student , course , student\_course

连接条件: student.id = student\_course.studentid , course.id = student\_course.courseid

1 select s.name , s.no , c.name from student s , student\_course sc , course c where

s.id = sc.studentid and sc.courseid = c.id ;

**备注: 以上需求的实现方式可能会很多, SQL写法也有很多，只要能满足我们的需求，查询出符合条件的记录即可。**

1. 事务

事务 是一组操作的集合，它是一个不可分割的工作单位，事务会把所有的操作作为一个整体一起向系统提交或撤销操作请求，即这些操作要么同时成功，要么同时失败。

为了解决上述的问题，就需要通过数据的事务来完成，我们只需要在业务逻辑执行之前开启事务，执行 完毕后提交事务。如果执行过程中报错，则回滚事务，把数据恢复到事务开始之前的状态。

注意： 默认MySQL的事务是自动提交的，也就是说，当执行完一条DML语句时，MySQL会立即隐式的提交事务。

1. 事务操作

### 控制事务一

1. 查看/设置事务提交方式
2. SELECT @@autocommit ;
3. SET @@autocommit = 0 ;

注意：上述的这种方式，我们是修改了事务的自动提交行为, 把默认的自动提交修改为了手动提交, 此时我们执行的DML语句都不会提交, 需要手动的执行commit进行提交。

### 控制事务二

1. 开启事务

1 START TRANSACTION 或 BEGIN ;

1. 提交事务

1 COMMIT;

1. 回滚事务

1 ROLLBACK;

转账案例：

1. -- 开启事务
2. start transaction 3
3. -- 1. 查询张三余额
4. select \* from account where name = '张三'; 6

7 -- 2. 张三的余额减少1000

8 update account set money = money - 1000 where name = '张三'; 9

10 -- 3. 李四的余额增加1000

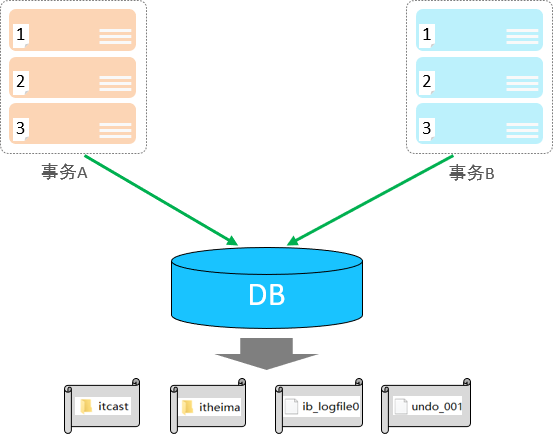
11 update account set money = money + 1000 where name = '李四'; 12

1. -- 如果正常执行完毕, 则提交事务
2. commit;
3. -- 如果执行过程中报错, 则回滚事务
4. -- rollback;
5. 事务四大特性ACID

原子性（Atomicity）：事务是不可分割的最小操作单元，要么全部成功，要么全部失败。 一致性（Consistency）：事务完成时，必须使所有的数据都保持一致状态。

隔离性（Isolation）：数据库系统提供的隔离机制，保证事务在不受外部并发操作影响的独立 环境下运行。

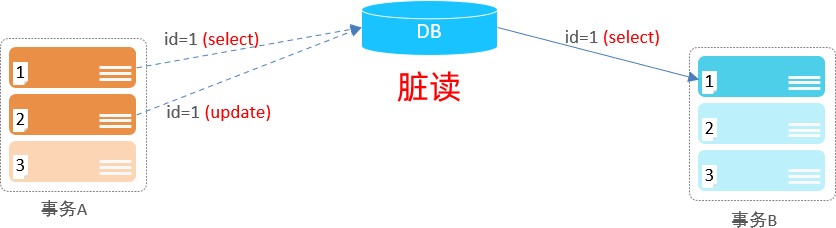
持久性（Durability）：事务一旦提交或回滚，它对数据库中的数据的改变就是永久的。



## 并发事务问题

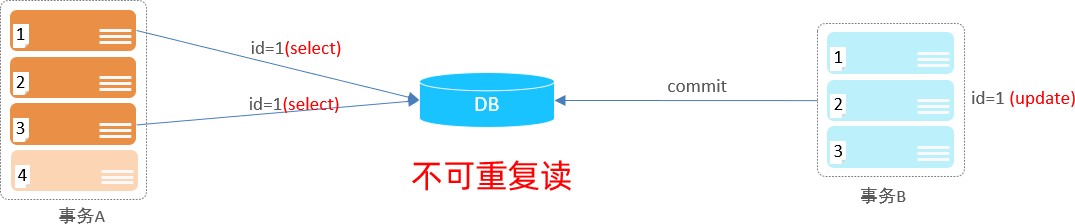
脏写：一个事务修改了另一个未提交的事务修改过的数据。

1. 赃读：一个事务读到另外一个事务还没有提交的数据。



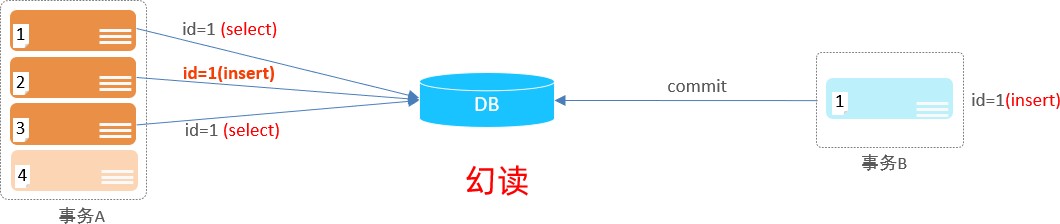
比如B读取到了A未提交的数据。

1. 不可重复读：一个事务先后读取同一条记录，但两次读取的数据不同，称之为不可重复读。



事务A两次读取同一条记录，但是读取到的数据却是不一样的。

1. 幻读：一个事务按照条件查询数据时，没有对应的数据行，但是在插入数据时，又发现这行数据 已经存在，好像出现了 "幻影"。



## 事务隔离级别

默认情况下，InnoDB在 REPEATABLE READ事务隔离级别运行，InnoDB使用 next-key 锁进行索引扫描，以防止幻读。

为了解决并发事务所引发的问题，在数据库中引入了事务隔离级别。主要有以下几种：

|  |  |  |  |
| --- | --- | --- | --- |
| **隔离级别** | **脏读** | **不可重复读** | **幻读** |
| **Read uncommitted** | √ | √ | √ |
| **Read committed** | × | √ | √ |
| **Repeatable Read(默认)** | × | × | √ |
| **Serializable** | × | × | × |

1. 查看事务隔离级别

1 SELECT @@TRANSACTION\_ISOLATION;

1. 设置事务隔离级别

1 SET [ SESSION | GLOBAL ] TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED |

READ COMMITTED | REPEATABLE READ | SERIALIZABLE }

注意：事务隔离级别越高，数据越安全，但是性能越低。

MySQL进阶篇

1. MySQL体系结构

客户端->连接层->服务层->引擎层->存储层

1. 建表时指定存储引擎

CREATE TABLE 表名(

字段1 字段1类型 [ COMMENT 字段1注释 ] ,

......

字段n 字段n类型 [COMMENT 字段n注释 ]

) ENGINE = INNODB [ COMMENT 表注释 ] ;

1. 查询当前数据库支持的存储引擎

show engines;

1. InnoDB （默认存储引擎）

特点：支持事务、外键、行级锁

1. 逻辑存储结构

表空间、段、区、页、行

1. MyISAM

特点：不支持事务，不支持外键、支持表锁，不支持行锁 、访问速度快

1. Memory

介绍 ：Memory引擎的表数据时存储在内存中的，由于受到硬件问题、或断电问题的影响，只能将这些表作为临时表或缓存使用。

特点：内存存放 、hash索引（默认）

1. 索引

索引是帮助MySQL高效获取数据的有序的数据结构。

1. 三个存储引擎都支持B+树索引，只有Memory支持hash索引
2. Hash只能用于对等比较，不能范围查询，且无法排序，但是查询效率高
3. 以查看当前数据库的INSERT、UPDATE、DELETE、SELECT的访问频次：

show global status like 'Com\_\_\_\_\_\_\_'; ()

1. 配置/etc/my.cnf添加以下两行：开启MySQL慢日志查询：

slow\_query\_log=1

设置慢日志的时间为2秒，SQL语句执行时间超过2秒，则记录慢查询日志

long\_query\_time=2

慢日志保存位置：/var/lib/mysql/localhost-slow.log (用 tail -f 跟踪）

1. show profiles 能够在做SQL优化时帮助我们了解时间都耗费到哪里去了。

查看当前MySQL是否支持：

SELECT @@have\_profiling ;

开启：

SET profiling = 1;

查看每一条SQL的耗时基本情况

show profiles;

查看指定query\_id的SQL语句各个阶段的耗时情况

show profile for query query\_id（上一条语句查到的id）;

1. 在select之前使用explain或desc 获取 MySQL 如何执行 SELECT 语句的信息

（可以检查索引是否命中）

1. 查询到的信息按行显示：

select \* from tb\_sku where id = 1\G;

1. 创建索引：（在tb\_sku表的sn列上创建）

create （unique)index idx\_sku\_sn on tb\_user(name,password) ;

1. 创建联合索引时要注意左边的索引应该考虑到在查询时尽可能使用到（最左前缀法则）最左前缀法则中指的最左边的列，是指在查询时，联合索引的最左边的字段(即是第一个字段)必须存在，与我们编写SQL时，条件编写的先后顺序无关。
2. 索引失效
   1. 联合索引中，出现范围查询(>,<)，范围查询右侧的列索引失效。当范围查询使用>= 或 <= 时，索引不会失效
   2. 不要在索引列上进行运算操作，否则索引将失效。
   3. 字符串类型字段使用时，不加引号，索引将失效。
   4. 如果仅仅是尾部模糊匹配，索引不会失效。如果是头部模糊匹配，索引失效。
   5. 用or分割开的条件， 如果or前的条件中的列有索引，而后面的列中没有索引，那么涉及的索引都不会被用到。
   6. 如果MySQL评估使用索引比全表更慢，则不使用索引。
3. SQL提示，是优化数据库的一个重要手段，简单来说，就是在SQL语句中加入一些人为的提示来达到优化操作的目的。

select \* from emp use index(idx\_name) where name=’tom’;

ignore index / force index

1. 覆盖索引是指 查询使用了索引，并且需要返回的列，在该索引中已经全部能够找到 。 （尽量使用覆盖索引，减少select \*，避免回表查询）
2. 前缀索引

create index idx\_xxxx on table\_name(column(n)) ;

当字段类型为字符串时，有时候需要索引很长的字符串，这会让索引变得很大，此时可以只将字符串的一部分前缀，建立索引

create index idx\_email\_5 on tb\_user(email(5));

查看前缀索引的选择性：

select count(distinct substring(email,1,5)) / count(\*) from tb\_user ;

使用了前缀索引必须要回表才能拿到需要的数据，即使是只查找索引本身所在的字段。所以，用了前缀索引，就用不了覆盖索引了。

1. 在业务场景中，如果存在多个查询条件，考虑针对于查询字段建立索引时，建议建立联合索引，而非单列索引。
2. 索引设计原则
   1. 针对于数据量较大，且查询比较频繁的表建立索引。
   2. 针对于常作为查询条件（where）、排序（order by）、分组（group by）操作的字段建立索引。
   3. 尽量选择区分度高的列作为索引，尽量建立唯一索引，区分度越高，使用索引的效率越高。
   4. 如果是字符串类型的字段，字段的长度较长，可以针对于字段的特点，建立前缀索引。
   5. 尽量使用联合索引，减少单列索引，查询时，联合索引很多时候可以覆盖索引，节省存储空间，避免回表，提高查询效率。
   6. 要控制索引的数量，索引并不是多多益善，索引越多，维护索引结构的代价也就越大，会影响增删改的效率。
   7. 如果索引列不能存储NULL值，请在创建表时使用NOT NULL约束它。当优化器知道每列是否包含NULL值时，它可以更好地确定哪个索引最有效地用于查询。
3. SQL优化-插入数据
4. 批量插入数据

Insert into tb\_test(...) values(1,'Tom'),(2,'Cat'),(3,'Jerry');

1. 手动控制事务

start transaction; (begin)

insert into tb\_test values(1,'Tom'),(2,'Cat'),(3,'Jerry');

insert into tb\_test values(4,'Tom'),(5,'Cat'),(6,'Jerry');

insert into tb\_test values(7,'Tom'),(8,'Cat'),(9,'Jerry');

commit;

1. 主键顺序插入，性能要高于乱序插入。
2. 大批量插入数据 ：load

-- 客户端连接服务端时，加上参数 -–local-infile

mysql –-local-infile -u root -p

-- 设置全局参数local\_infile为1，开启从本地加载文件导入数据的开关

select @@local\_infile;

set global local\_infile = 1;

-- 执行load指令将准备好的数据，加载到表结构中

先在mysql中创建tb\_user表

load data local infile '/root/load\_user\_100w\_sort.sql' into table tb\_user

fields terminated by ',' lines terminated by '\n' ;

1. SQL优化-主键优化

在InnoDB存储引擎中，表数据都是根据主键顺序组织存放的，主键顺序插入的性能是要高于乱序插入的

1. 乱序插入会导致“页分裂”比较耗费性能
2. 删除时会导致页合并优化空间，合并页的阈值，可以自己设置，在创建表或者创建索引时指定。

当删除一行记录时，实际上记录并没有被物理删除，只是记录被标记（flaged）为删除并且它的空间变得允许被其他记录声明使用。当页中删除的记录达到 MERGE\_THRESHOLD（默认为页的50%），innoDB会开始寻找最靠近的页（前或后）看看是否可以将两个页合并以优化空间使用。

1. 主键索引设计原则
2. 满足业务需求的情况下，尽量降低主键的长度。
3. 插入数据时，尽量选择顺序插入，选择使用auto\_increment自增主键。尽量不要使用UUID做主键或者是其他自然主键，如身份证号。
4. 业务操作时，避免对主键的修改。
5. order by优化

MySQL的排序，有两种方式：

Using filesort：在排序缓冲区sort buffer中完成排序操作

Using index：通过有序索引顺序扫描直接返回有序数据，操作效率高。

1. 根据排序字段建立合适的索引，多字段排序时，也遵循最左前缀法则。 (order后的字段顺序最好和建立联合索引时的顺序一致，不一致就会filesort）
2. 尽量使用覆盖索引。 （select内容都在索引中）
3. 多字段排序, 一个升序一个降序，此时需要注意联合索引在创建时的规则。

create index idx\_user\_age\_phone\_ad on tb\_user(age asc ,phone desc);

1. 如果不可避免的出现filesort，大数据量排序时，可以适当增大排序缓冲 区大小 sort\_buffer\_size(默认256k)。
2. group by优化
3. 在分组操作时，可以通过索引来提高效率。
4. 分组操作时，索引的使用也是满足最左前缀法则的。
5. limit优化

例如查询2000000条到2000010条（limit 2000000,10）

优化思路: 一般分页查询时，通过创建 覆盖索引 能够比较好地提高性能，可以通过覆盖索引加子查询形式进行优化。

explain select \* from tb\_sku t , (select id from tb\_sku orde r by id

limit 2000000,10) a where t.id = a.id;

1. count优化

如果说要大幅度提升InnoDB表的count效率，主要的优化思路：自己计数(可以借助于redis这样的数 据库进行,但是如果是带条件的count又比较麻烦了)。

按照效率排序的话，count(字段) < count(主键 id) < count(1) ≈ count(\*)，所以尽

量使用 count(\*)。

1. update优化

InnoDB的行锁是针对索引加的锁，不是针对记录加的锁 ,并且该索引不能失效，否则会从行锁 升级为表锁 。 （回表时就会变为表锁，此时并发开启多个事务时，性能将大大降低）

1. 视图

是一种虚拟存在的表(select查询时的表)。视图中的数据并不在数据库中实际存在

1. 创建

create or replace view stu\_v\_1 as select id,name from student where id <= 10 with cascade check option;

1. 查询

查看创建视图语句：SHOW CREATE VIEW 视图名称;

查看视图数据：SELECT \* FROM 视图名称 ...... ;

1. 修改

create/alter [or replace] view 视图名称[(列名列表)] as select语句 [ with

[ cascade | local ] check option ]

1. 删除

drop view [IF EXISTS] 视图名称 [,视图名称] ...

1. mysqlL允许基于另一个视图创建视图

也可以通过视图来插入、更新基表数据

不符合视图创建条件的数据不会插入视图但是会插入基表中

怎样做到满足条件才能插入基表中？

答案是使用with casecaded/local check option

1. casecaded 级联

比如，v2视图是基于v1视图的，如果在v2视图创建的时候指定了检查选项为 cascaded，但是v1视图 创建时未指定检查选项。 则在执行检查时，不仅会检查v2，还会级联检查v2的关联视图v1。

1. local 本地

比如，v2视图是基于v1视图的，如果在v2视图创建的时候指定了检查选项为 local ，但是v1视图创 建时未指定检查选项。 则在执行检查时，只会检查v2，不会检查v2的关联视图v1。

1. 视图的更新

要使视图可更新，视图中的行与基础表中的行之间必须存在一对一的关系。如果视图包含以下任何一 项，则该视图不可更新：

A. 聚合函数或窗口函数（SUM()、 MIN()、 MAX()、 COUNT()等）

B. DISTINCT

C. GROUP BY

D. HAVING

E. UNION 或者 UNION ALL

1. 视图作用
   1. 简单（不必每次操作都指定全部条件）
   2. 安全（通过视图只能操作所能看到的数据）
   3. 数据独立（屏蔽真实表结构变化带来的影响）
2. 存储过程\_基本语法

create procedure pro1( )

begin

select \* from tab1;

end;

调用：call pro1( );

1. 查看存储过程
   1. 查看存储过程及状态信息：

select \* from information\_schema.ROUTINEES where ROUTINE\_SCHEMA = ‘数据库名’;

* 1. 查看定义信息：show create procedure pro1;

1. 删除

drop procedure if exists pro1;

1. 在命令行中，执行创建存储过程的SQL时，需要通过关键字 delimiter 指定SQL语句的 结束符。

delimiter $$

begin pro...

select....;(未结束)

end$$

delimiter ;

1. 变量

在MySQL中变量分为三种类型: 系统变量、用户定义变量、局部变量。

查看

show session/global variables like ‘...’;

select @@系统变量名;（查看单个）

设置

set session/global 系统变量名 = 值 ;

1. 注意：

如果没有指定SESSION/GLOBAL，默认是SESSION，会话变量。

mysql服务重新启动之后，所设置的全局参数会失效，要想不失效，可以在/etc/my.cnf 中配置。

全局变量(GLOBAL): 全局变量针对于所有的会话。

会话变量(SESSION): 会话变量针对于单个会话，在另外一个会话窗口就不生效了。

1. 用户定义变量
2. set @var\_name ：= expr [, @var\_name = expr] ... ;
3. select count(\*) into 变量名 from tab1;（查询结果作为变量的值）
4. 局部变量

根据需要定义的在局部生效的变量，访问之前，需要DECLARE声明。范围是在其内声明的BEGIN ... END块。

create procedure pro2( )

begin

declare var1 int default 0;

select count(\*) into var1 from student;

select var1;

end;

1. 参数

类型：in（输入）、out（输出）、inout（既可以作为输入也可以作为输出）

create procedure pro3(in score int, out result varchar(10))

begin

if score>85 then

set result:=’优秀’;

elseif score>60 then

set resoult:=’及格’;

else

set resoult:=’不及格’;

end if;

end;

call pro3(65,@resoult);

select @resoult;

1. case(看例题）
2. while(看例题）
3. repeat(看例题）
4. loop(看例题）
5. 游标

游标（CURSOR）是用来存储查询结果集的数据类型 , 在存储过程和函数中可以使用游标对结果集进 行循环的处理。使用：open fetch close

1. 条件处理程序

条件处理程序（Handler）可以用来定义在流程控制结构执行过程中遇到问题时相应的处理步骤。

1. 存储函数

存储函数（function）是有返回值的存储过程，存储函数的参数只能是IN类型的。

1. 触发器

触发器是与表有关的数据库对象，指在insert/update/delete之前(BEFORE)或之后(AFTER)，触发并执行触发器中定义的SQL语句集合。现在触发器还 只支持行级触发，只支持行级触发，不支持语句级触发。

create trigger trigger\_name

before/after inser/update/delete

on tbl\_name for each row -- 行级触发器

begin

insert into 日志表 //记录修改内容

end;

查看触发器：

show triggers;

删除触发器：

drop trigger schema\_name.trigger\_name;

1. 锁

锁是计算机协调多个进程或线程并发访问某一资源的机制。

1. 全局锁

对数据库进行进行逻辑备份之前，先对整个数据库加上全局锁，一旦加了全局锁之后，其他的DDL、 DML全部都处于阻塞状态，但是可以执行DQL语句，也就是处于只读状态，而数据备份就是查询操作。

1. 数据备份语法

加全局锁 flush tables with read lock ;

数据备份 mysqldump -uroot –p1234 itcast > itcast.sql

释放锁 unlock tables ;

在InnoDB引擎中，我们可以在备份时加上参数 --single-transaction 参数来完成不加锁的一致 性数据备份

1. 表级锁（表锁、元数据锁、意向锁）
2. 对于表锁，分为两类：

表共享读锁（read lock）

表独占写锁（write lock）

语法：

加锁：lock tables 表名... read/write。

释放锁：unlock tables /或客户端断开连接 时自动释放。

结论: 读锁不会阻塞其他客户端的读，但是会阻塞写。写锁既会阻塞其他客户端的读，又会阻塞 其他客户端的写但是不阻塞自身的读和写。

1. 元数据锁

MDL加锁过程是系统自动控制，无需显式使用，在访问一张表的时候会自动加上。MDL锁主要作用是维护表元数据的数据一致性，在表上有活动事务的时候，不可以对元数据进行写入操作。**为了避免DML与DDL冲突，保证读写的正确性。**这里的元数据，可以简单理解为一张表的表结构。**也就是说，一张表涉及到未提交的事务时，是不能够修改这张表的表结构的**。

当执行select、insert、update、delete等语句时，添加的是元数据共享锁（shared\_read /shared\_write），之间是兼容的。

而alter语句添加的exclusive锁与元数据读写锁都是互斥的

1. 意向锁

为了避免DML在执行时，加的行锁与表锁的冲突，在InnoDB中引入了意向锁，使得表锁不用检查每行数据是否加锁，使用意向锁来减少表锁的检查。

意向共享锁(IS): 由语句select ... lock in share mode添加 。 与 表锁共享锁

(read)兼容，与表锁排他锁(write)互斥。

意向排他锁(IX): 由insert、update、delete、select...for update添加 。与表锁共

享锁(read)及排他锁(write)都互斥，意向锁之间不会互斥。

一旦事务提交了，意向共享锁、意向排他锁，都会自动释放。

可以通过以下SQL，查看意向锁及行锁的加锁情况：

select object\_schema,object\_name,index\_name,lock\_type,lock\_mode,lock\_data from performance\_schema.data\_locks;

1. 行级锁（基于索引）（行锁，间隙锁，临键锁）
2. 行锁

共享锁（S）：允许一个事务去读一行，阻止其他事务获得相同数据集的排它锁。

排他锁（X）：允许获取排他锁的事务更新数据，阻止其他事务获得相同数据集的共享锁和排他锁。（S和S兼容，X和S、X都互斥）

没有索引，行锁会升级为表锁

1. 间隙锁&临键锁

索引上的等值查询(唯一索引)，给不存在的记录加锁时, 优化为间隙锁 。

索引上的等值查询(非唯一普通索引)，向右遍历时最后一个值不满足查询需求时，next-key lock 退化为间隙锁。

索引上的范围查询(唯一索引)--会访问到不满足条件的第一个值为止。

间隙锁唯一目的是防止其他事务插入间隙。间隙锁可以共存，一个事务采用的间隙锁不会阻止另一个事务在同一间隙上采用间隙锁。

1. InnoDB引擎

表空间

每张表都会有一个表空间（xxx.ibd），一个mysql实例可以对应多个表空间

段

数据段就是B+树的叶子节点， 索引段即为B+树的 非叶子节点。

区

每个区的大小为1M。 默认情况下， InnoDB存储引擎页大小为16K， 即一

个区中一共有64个连续的页。

页

页，是InnoDB 存储引擎磁盘管理的最小单元，每个页的大小默认为 16KB。为

行

InnoDB 存储引擎数据是按行进行存放的

在行中，默认有两个隐藏字段：

Trx\_id：每次对某条记录进行改动时，都会把对应的事务id赋值给trx\_id隐藏列。

Roll\_pointer：每次对某条引记录进行改动时，都会把旧的版本写入到undo日志中，然后这个

隐藏列就相当于一个指针，可以通过它来找到该记录修改前的信息。

1. 架构

MySQL5.5 版本开始，默认使用InnoDB存储引擎，它擅长事务处理，具有崩溃恢复特性，在日常开发中使用非常广泛。

内存结构：

缓冲池：经常使用到的数据

更改缓冲区：在执行DML语句时，如果这些数据Page

没有在Buffer Pool中，不会直接操作磁盘，而会将数据变更存在更改缓冲区 Change Buffer

中，在未来数据被读取时，再将数据合并恢复到Buffer Pool中，再将合并后的数据刷新到磁盘中。

自适应hash索引：如果观察到在特定的条件下hash索引可以提升速度，则建立hash索引，称之为自适应hash索引。（自动）

日志缓冲区：用来保存要写入到磁盘中的log日志数据，定期刷新到磁盘中。

磁盘结构：

1. 后台线程

在InnoDB的后台线程中，分为4类，分别是：Master Thread 、IO Thread、Purge Thread、Page Cleaner Thread。

1. 事务原理

原子性、一致性、持久化，实际上是由InnoDB中的两份日志来保证的，一份是redo log日志，一份是undo log日志。

而持久性是通过数据库的锁，加上MVCC来保证的。

redo log：重做日志，记录的是事务提交时数据页的物理修改，是用来实现事务的持久性。

undo log ：回滚日志，用于记录数据被修改前的信息 , 作用包含两个 : 提供回滚(保证事务的原子性) 和MVCC(多版本并发控制) 。

1. MVCC
   1. 当前读

读取的是记录的最新版本，读取时还要保证其他并发事务不能修改当前记录，会对读取的记录进行加锁。对于我们日常的操作，如：select ... lock in share mode(共享锁)，select ...

for update、update、insert、delete(排他锁)都是一种当前读。

* 1. 快照读

简单的select（不加锁）就是快照读，快照读，读取的是记录数据的可见版本，有可能是历史数据，

不加锁，是非阻塞读。

* 1. MVCC

多版本并发控制。指维护一个数据的多个版本，使得读写操作没有冲突，需要依赖于数据库记录中的三个隐式字段、undo log日志、readView（读视图）是 快照读 。

1. MySQL管理
   1. mysql
   2. mysqladmin
   3. mysqlbinlog
   4. mysqlshow（查找）
   5. mysqldump（备份）

mysqldump -uroot -p1234 db01 > db01.sql

* 1. mysqlimport/source

source /root/xxxxx.sql

运维篇

1. 错误日志 /var/log/mysql.log

记录了当 mysqld 启动和停止时，以及服务器在运行过程中发生任何严重错误时的相关信息。当数据库出现任何故障导致无法正常使用时，建议首先查看此日志。/var/log/mysql.log

show variables like '%log\_error%';

1. 二进制日志

二进制日志（BINLOG）记录了所有的 DDL（数据定义语言）语句和 DML（数据操纵语言）语句，但

不包括数据查询（SELECT、SHOW）语句。

作用：①. 灾难时的数据恢复；②. MySQL的主从复制。在MySQL8版本中，默认二进制日志是开启着

的，涉及到的参数如下：

show variables like '%log\_bin%';

查看：mysqlbinlog [ 参数选项 ] logfilename

1. 查询日志 /var/lib/mysql/mysql\_query.log

查询日志中记录了客户端的所有操作语句，而二进制日志不包含查询数据的SQL语句。

1. 慢查询日志

慢查询日志记录了所有执行时间超过参数 long\_query\_time 设置值并且扫描记录数不小于

min\_examined\_row\_limit 的所有的SQL语句的日志，默认未开启。long\_query\_time 默认为

10 秒，最小为 0， 精度可以到微秒

1. 主从复制

主从复制是指将主数据库的 DDL 和 DML 操作通过二进制日志传到从库服务器中，然后在从库上对这

些日志重新执行（也叫重做），从而使得从库和主库的数据保持同步。

过程：

主库在事务提交时，记录binlog->从库读取写入自己的relay Log->从库重做中继日志中的事件。

搭建：

配置my.cnf

1. 分库分表

当数据量变得非常大时，只在一个主机上管理这些数据非常吃力，性能跟不上。

垂直分库：把不同的表分到不同的主机上

垂直分表：把一张表上不同的列分到不同的主机上

水平分库：所有数据库的表都一样，表内数据不一样

水平分表：将一个表的数据拆分到多个表中，表结构一样，数据不一样

1. 实现技术
   1. shardingJDBC：自行编码配置，只支持java，性能高
   2. MyCat：智能中间件，多种语言，性能稍低
2. MyCat分片规则
   1. 范围分片（如id1-500,501-1000）
   2. 取模分片 （如id%3的3个不同的值分到不同的表）
   3. 一致性hash分片
   4. 枚举分片 （如status为0,1,2）
   5. 应用指定算法 （去字符串子串）
   6. 固定分片hash算法 （类似十进制求模运算）
   7. 字符串hash解析算法 （截取字符串中的指定位置的子字符串, 进行hash算法， 算出分片。
   8. 按天分片算法
   9. 自然月分片
3. MyCat原理

在MyCat中，当执行一条SQL语句时，MyCat需要进行SQL解析、分片分析、路由分析、读写分离分析

等操作，最终经过一系列的分析决定将当前的SQL语句到底路由到那几个(或哪一个)节点数据库，数据

库将数据执行完毕后，如果有返回的结果，则将结果返回给MyCat，最终还需要在MyCat中进行结果合

并、聚合处理、排序处理、分页处理等操作，最终再将结果返回给客户端。

1. 读写分离

把对数据库的读和写操作分开,以对应不同的数据库服务器。主数据库提供写操作，从数据库提供读操作，这样能有效地减轻单台数据库的压力。

1. 双主双从

一个主机 Master1 用于处理所有写请求，它的从机 Slave1 和另一台主机 Master2 还有它的从

机 Slave2 负责所有读请求。当 Master1 主机宕机后，Master2 主机负责写请求，Master1 、

Master2 互为备机。

B树和B+树的区别

https://blog.csdn.net/FeiChangWuRao/article/details/124311192?ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522167115643016782395342498%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=167115643016782395342498&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~top\_click~default-2-124311192-null-null.142^v68^control,201^v4^add\_ask,213^v2^t3\_control2&utm\_term=B%E6%A0%91%E5%92%8CB%2B%E6%A0%91%E7%9A%84%E5%8C%BA%E5%88%AB&spm=1018.2226.3001.4187