基本概念

1.RDBMS:关系型数据库管理系统（数据存储在表里，表之间可产生关联，执行和维护数据库任务的软件）

2.关于MySQL：oracle公司产品，三大版（社区版（开源、免费的）、企业版（标准版、企业版（支持增量备份）、电信级别版（精确到99.999%+日志管理+监视器）））

mysql注释 ctrl+/或者/\* \*/ 取消注释:ctrl+shift+/

3数据库对象：[表](#表) [索引](#索引) [视图](#视图) [存储过程](#存储过程),[复合语句](#复合语句)，[函数](#常用函数),[触发器](#触发器),[游标](#游标) [异常处理](#异常处理)，[事务](#事务) 性能优化

MySQL的数据类型

1 数值类型

A整数

tinyint（1字节8bit 0-255或-128~127）smallint（2字节0-65535或-32768~32767）

mediumint （3个字节）int/integer（4个字节）bigint(8个字节)

B小数float（0-23bit）/double(0-53bit)/double(p,s)(p:表精度 s:表留的小数位数)

C Bit类型（每一个位存放一个0或1）

2字符串

Char(n) 1-255固定长度varchar(n)/tinytext/ 1-255可变长度

text(2字节)Mediumtext（4字节） longtext(8字节)

Binary (n) 1字节/Blod (n) 2字节二进制数据固定长eg 存储图片

3日期

Date 1000-1-1到9999-12-31

Time 838：59:59到838：59：59

datatime 1000-1-1 00:00:00到9999-12-31 23:59：

timestamp时间戳1970-01-01 00:00:00到2037-12-31用来记录操作日志

year 1901到2155

Navicat（数据库管理软件）

新建连接：主机（localhost或127.0.0.1）或IP地址端口（3306）用户名密码

创建数据库🡪创建表🡪添加数据

对MySQL的操作分两种:1 使用Navicat 2使用SQL语句

--创建数据库，如果不存在就创建，存在则不创建

create database if not exists student;

-- 删除数据库(mysql不允许删除系统数据库) drop database if exists student;

-- 修改数据库alter database student character set utf8;

--表的操作：创建表-- 使用数据库，切换

约束（constraint）：目的是为了维护数据的完整性

A 主键约束：可唯一确定某行数据的一列或者某些列，数据不能为空(primary key)

（主键只有一个可以是一个子段也可以是多个子段组成的联合主键）

B唯一约束：非主键列、也不能重复的数据列constraint约束名unique(列名)

C外键约束：应用其他表的主键的某些列，使得当前表列的数据范围不得超出引用列的范围

Eg use student;

create table if not exists stu(

id int not nullauto\_increment(自增长)primary key,-- 主键约束

name varchar(20) not null,

birth datetime或timestamp defaultcurrent\_timestamp ,

age int,

major varchar(10),

classid int,

constraint unique\_name unique(name),-- 唯一约束

constraint fk foreign key(classid) references class(id)—外键约束

)

修改表

约束操作

添加约束:alter table 表 add constraint 约束名 约束描述

添加外键约束:alter table 外建表 add constraint 约束名 foreign key(外键表的列名) references 主键表 (主键表的列名)

添加唯一约束:alter table 表 add constraint 约束名 unique(列名)

-- alter table stu drop primary key;删除主键约束

-- alter table stu drop foreign key fk;删除外键约束

-- alter table stu drop index unique\_age;删除唯一约束

列操作

-- 添加列

alter table stu add column address varchar(50) first;

alter table stu add column grade double after id;

-- 删除列

alter table stu drop column address;

-- 修改列

alter table stu modify column major int;

alter table stu change column major major1 int;—修改列名

alter table stu alter column name set default ‘tom’;--设置列的默认值

数据操作

添加数据

-- 1所有列都插入数据

insert into emp values(1001,'lili',20,66666.88,'香港',2001);

insert into emp set id=1007,name='mike',age=22,salary=55555,address='天津',deptid='2001';

replace into emp values(1006,'tom',28,55555.5,'广州',2001);

注：replace into若插入的主键已在，则删除之再添加；若不存在就添加相当于insert

-- 2部分列插入数据可指定要插入的列的列名(包含所有非空的列也包括主键)

insert into emp(id,name,age) values(1003,'kelly',30);

insert into emp set id=1008,name='mike',deptid='2001';

replace into emp(id,name,age) values(1006,'tom',28);

-- 3插入多条数据

insert into emp values

(1004,'john',25,55555.5,'北京',2001),

(1005,'james',25,599995.5,'上海',2001),

(1006,'tomy',25,55555.5,'广州',2001);

修改更新数据set语句，指定要更新的列及其值，可以修改主键，但是新的主键不是已存在的主键update 表名 set 列1=值1,列2=值2....... where ....

update stu set name='hua',age=20 where id=6;

update emp set id='1009',age=25 where id=1005;

删除数据

1 delete删除所有数据或若干行数据

delete from emp;-- 慎用delete from emp where id=1009;

2 truncate用于清空表，先删除(drop)表再创建(create),不用where,速度比delete快

truncate table emp;-- 慎用(测试模拟等使用)

查询

select \* from 表名 where ... group by ...having.... order by asc|desc limit

-- 查询所有行和列即所有数据，\*代表所有列 eg select \* from emp;

-- 查询部分列 eg select id,name,address from emp;

-- 使用别名eg select id as 员工ID,name as 员工姓名 from emp;

-- 查询选定行(where)

--A 逻辑运算符默认优先,但是有括号先执行括号里的

-- not(!)>and(&&同时成立)>or(|| 有一个成立即可)

select \* from emp where id=1002 and name='bob';

select \* from emp where id=1002 or id=1003;

select \* from emp where not id=1002;-- 除了1002之外的所有数据

select \* from emp where id=1002 and name='bob' or id=1003;

select \* from emp where (name='bob' or id=1002) and id=1003;-- 返回空结果

--B 比较运算符=,>,<,>=,<=,!=或<>

select \* from emp where salary>88888;

select \* from emp where deptid<>2001;-- 不等于2001

--C between..A..and...B... 在A和B之间

select \* from emp where age between 25 and 50;

select \* from emp where age>=30 and age<=50;

--D like 像....一样相似不精确查询

-- %通配符,代表任意位数的字符串, \_下划线,代表任意一位字符

select \* from emp where name like 'l%';-- 所有以l开头的名字的记录

select \* from emp where name like '%y';-- 所有以y结尾的名字的记录

select \* from emp where name like '%m%';-- 所有含有m的名字的记录

select \* from emp where name like 'to\_';-- 所有以to开头再带一位的名字的记录

-- in语句，用于集合条件

select \* from emp where address='香港' or address='上海';

select \* from emp where address in('香港','上海');

-- limit语句，用于分页查询

select \* from emp limit 2,5;-- 2表跳过前2条数据，5表查询要总的数据行数

select \* from emp limit 0,5;-- 查询前5条数据

--常用函数

-- 字符串函数

-- 截取字符串left(str,n)/right/substring

select left(name,2) from emp where id=1007;-- 取左边2位

select right(name,3) from emp where id=1007;-- 取右边3位

select substring(name,2,2) from emp where id=1002;-- 取从第2位长为2

-- reverse字符串逆向输出

select reverse(name) from emp where id=1003;

-- lower/upper大小写

-- 正则表达式作为查询条件 regexp

select \* from emp where name regexp '^m';

select \* from emp where name regexp 'm.\*';-- .表任意字符 \*表出现0到n次

-- 日期函数

-- date获取年月日

select date('2016-12-22 14:14:20');

-- datediff计算日期的差值(天数)

select datediff('2016-12-22 14:14:20','2008-02-22 20:20:22');

--数学函数

select salary from emp where id=1004;-- 月薪55555.5

select salary\*12 from emp where id=1004;-- 年薪666666

-- floor(x)向下取整 ceiling(x)向上取整 round(x)四舍五入

select floor(salary) from emp where id=1004;-- 55555

select ceiling(salary) from emp where id=1004;-- 55556

select round(salary) from emp where id=1004;-- 55556

-- 聚合函数：最值(max/min) avg sum count

select max(salary) from emp;

select min(salary) from emp;

select avg(salary) from emp;

select round(avg(salary)) from emp;

select sum(salary) from emp;

select count(\*) from emp;-- 返回满足条件的数据的数目

-- 分组

-- 先按照地址分组以及年龄大于30的条件筛选再选取地址和平均工资

(1) （SQL语句执行顺序从右往左：判断条件再选取数据）

select address,avg(salary) from emp where age>=30 group by address;

(2) Group by子句必须放在where语句的后面

(3) 没有出现在Group by子句中的列是不能放到select语句后的列名列表中的（聚合函数除外）。

  1) 错误：select FAge,FSalary from T\_Employee group by Fage。

  2) 正确：select Fage,Avg(FSalary) from T\_Employee group by Fage。

-- having对分组进行再筛选

(1) 在where中不能使用聚合函数，须使用Having，Having要位于Group by之后。

   select Fage,Count(\*) as 人数 from T\_Employee Group by FAge Having Count(\*)>1

1. 注意Having中不能使用为参数分组的列，Having不能代替where,作用不一样，Having是对组进行过滤。限制结果集行数

select address,avg(salary) from emp where age>=25 group by address having address <> '北京';

-- 排序(默认为升序)order by

select \* from emp order by age;-- 升序

select \* from emp order by age desc;-- 降序

select \* from emp order by address,age;-- 多列排序

-- 连接（内联和外联:联合查询）和子查询

-- 查询员工id,姓名,部门名称

select e.id,e.name,d.name from emp e,dept d where e.deptid=d.id order by d.name;

-- 内联接（满足条件的有关联的）

select e.id,e.name,d.name from emp e inner join dept d on e.deptid=d.id;

-- 左外联接（内连接数据加上若不满足on的条件的左边表emp e的数据e.id,e.name）

select e.id,e.name,d.name from emp e left outer join dept d on e.deptid=d.id;

-- 右外联接（内连接数据加上若不满足on的条件的右边表dept d的数据d.name ）

select e.id,e.name,d.name from emp e right outer join dept d on e.deptid=d.id;

-- 交叉连接

select e.id,e.name,d.name from emp e cross join dept d;

-- 自连接查询出bossid为1001的所有员工

select e2.\* from emp e1,emp e2 where e1.id=1001 and e2.bossid=e1.id;

-- 子查询,需跟在比较符之后，查询出跟id为1001同一个bossid的所有员工

select e1.\*from emp e1,emp e2 where e1.bossid=e2.bossid and e2.id=1001;

select \* from emp where bossid=(select bossid from emp where id=1001);

-- =等号，返回满足一个条件的数据

select \* from emp

where bossid=(select bossid from emp where id=1001);

-- in和not in 返回满足多个条件的数据

select \* from emp

where bossid in(select bossid from emp where id=1001);

-- exists和not exists 返回的不是数据，返回true或false

-- id=1008的数据存在，则返回所有数据，若不存在就返回空

select \* from emp

where exists(select \* from emp where id=1008);

-- select bossid from emp where id>1002

-- 结果(bossid)result：1001 1002 1003 1004

-- select \* from emp where bossid>all(....)

-- 结果为空因为表中bossid都比result大的数据没有

select \* from emp

where bossid>all(select bossid from emp where id>1002);

-- select bossid from emp where id>1002

-- 结果(bossid)result：1001 1002 1003 1004

-- select \* from emp where bossid>any(....)

-- 结果集因为表中bossid有比result中的最小的bossid大的数据

select \* from emp

where bossid>any(select bossid from emp where id>1002);

-- distinct去掉重复数据

select \* from emp where bossid>1002;

select distinct bossid from emp where id>1002;

索引

1索引--提高数据的查找的效率不使用\*,like,比较符

2 索引类型

A主键索引：此索引是写在表中唯一且不为空的列上

B外键索引、常规索引（写在任意列上）

C唯一索引：写在唯一的列上，可为空值

D为某些关键词或字加上索引

-- 创建索引

create [unique]index 索引名 on 表名(列1,.....)

-- 常规索引 emp表的salary上创建索引

create index index\_1 on emp(salary);

-- 唯一索引不允许索引列的值重复eg index\_2

create unique index index\_2 on emp(name);

-- 全文索引

create fulltext index index\_3 on emp(address);

-- 查看索引 主键索引在创建主键时即建立，所以结果为4个索引

show indexes from emp;-- 显示emp表上的所有索引

-- 删除索引

drop index index\_1 on emp;

视图

-- 视图：虚拟表，存放某个查询语句的结果集,保护表和表中的数据

-- 限制用户对于表和表中的数据直接操作,也提高数据访问效率eg 视图v3

create view v1 as select id,name,salary,address

from emp where id>1003;

create view v2 as

select id 列1,name 列2,salary 列3,address 列4

from emp where id>1003;

create view v3 as

select e.id 列1,e.name 列2,e.salary 列3,e.address 列4

from emp e inner join dept d on e.deptid=d.id;

-- 查询视图select \* from v2;select \* from v3;

-- 操作视图(更新，删除，添加是有条件的)

-- 能操作视图的条件视图查询中不能有聚合函数(max min avg count sum)

-- 也不能有distinct、子查询

-- 删除视图数据，也会删除底层表（基表数据来源表）中的数据

delete from v2 where 列1=1001;

-- 添加视图数据，也会添加到底层表（基表数据来源表）中的数据

insert into v2 values(1011,'haity',44444,'海口');

存储过程

存储过程:

#声明局部变量（只能在当前存储过程中使用）

DECLARE i INT DEFAULT 0;#声明整型的变量i并赋初值为0;

DECLARE address VARCHAR(50) DEFAULT '无锡市';

# 当前整个连接中有效: SET @x=100;

SET i =100;SET address ='上海市'; #赋值

1）不带参数的存储过程

CREATE PROCEDURE getEmp() # 创建存储过程

BEGIN

SELECT \* from emp;

SELECT \* from dept;

END;

CALL getEmp();# 调用存储过程

2）带参数的存储过程

CREATE PROCEDURE calcSalary(INempId INT, OUTempName varchar(20), INOUTempSalary INT)

BEGIN#根据员工id查询员工姓名

SELECT emp\_name into empName from emp where emp\_id = empId;

SELECT emp\_salary+empSalary into empSalary from emp where emp\_id = empId;

END;

SET @empId=1007;

SET @empSalary = 500;

callcalcSalary(@empId,@empName,@empSalary);

select @empName,@empSalary;//若有输出参数 call 存储过程名(@?.....) ?代表一个变量

复合语句：分支语句和循环语句

1）IF-ELSE-THEN 语句

IF i> 90 THEN SET result ='优';

ELSE IF i> 80 THEN SET result = '良';

ELSE SET result = '及格';

END IF;

END IF;

2) CASE-WHEN 语句

SELECT emp\_address AS 目的地,

CASE emp\_address

WHEN '海口市' THEN '清补凉'

WHEN '重庆市' THEN '火锅'

WHEN '贵阳市' THEN '茅台'

END AS 著名小吃

FROM emp;

循环语句

1）WHILE

Select emp\_salary into salary from emp where emp\_id=1007;

WHILE salary < 1000000 DO # WHILE...DO

SET salary = salary + 10000;

END WHILE;

Update emp set emp\_salary = salary where emp\_id=1007;

2）LOOP

loop1 : LOOP

SET salary = salary -5000;

IF salary <= 500000 THEN

leave loop1; #跳出循环，结束循环

END IF;

END LOOP;

updateemp set emp\_salary = salary where emp\_id=1007;

loop1 : LOOP

SET salary = salary + 5000;

IF salary <= 1000000 THEN

ITERATE loop1; # 如果salary <1000000 循环就继续

END IF;

leave loop1;

END LOOP;

Update emp set emp\_salary = salary where emp\_id=1007;

3）REPEATE

SET i=1;

REPEAT SET sum = sum +i; SET i = i +1; UNTIL i>100

END REPEAT;

异常处理

1）处理异常

BEGIN

# DECLARE EXIT HANDLER FOR SQLSTATE '23000' SET @x1=1; 出错就退出

# 1 mysql\_error\_code 4 位整数，MySQL特有的错误编码

# 2 SQLSTATE 标准(通用)的错误代码 5位的整数

DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x1=1; #出错之后也继续

SET @x=1;insert into emp (emp\_id,emp\_name,emp\_salary) values(2016,'slt',5000);

SET @x=2; insert into emp (emp\_id,emp\_name,emp\_salary) values(2016,'slt',5000); SET @x=3;

END

2）抛出自定义异常

BEGIN DECLARE salary INT; SET salary = 30000;

IF salary > 20000 THEN

SIGNAL SQLSTATE '45000' #未处理的用户自定义异常

SET MESSAGE\_TEXT = '工资不可以高于两万';

END IF;

END

游标

BEGIN

DECLARE empId INT; DECLARE empSalary INT;

DECLARE done INT DEFAULT FALSE; # 1 声明游标

DECLARE cur1 CURSOR FOR SELECT emp\_id,emp\_salary FROM emp;

#当游标抛出没有数据异常时候，将循环结束标志设置为true

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=true;

OPEN cur1; # 2 打开游标

update\_loop : LOOP # 3 使用数据

IF done THEN leave update\_loop; END IF;

FETCH cur1 into empId,empSalary; # 一次获取一条游标数据

UPDATE emp SET emp\_salary = empSalary+5000 where emp\_id=empId;

END LOOP update\_loop;

CLOSE cur1; # 4 关闭游标

END

触发器:

1.在执行insert，update，delete等SQL语句时会自动执行的一组SQL语句，.分类：前触发器before和后触发器after（实现备份）

2.drop trigger if exists触发器名

Insert 之前 null 之后 insert的数据Update 之前旧值之后新值

Delete 之前旧值之后 null

Create trigger tr\_insert\_emp before insert on emp;

For each row//创建触发器，只在emp表执行插入每一个记录时触发

Begin

If new.salary<0 then//抛出异常

signal sqlstate‘45000’set message\_text=’工资不能为负数’

end if

end

insert into emp(id,name,salary) values(1066,’may’,-111);//验证触发器

Create trigger tr\_insert\_emp after delete on emp; //在delete之后触发

For each row

Begin //实现备份

Insert into emp\_copy(id,name,salary) values(old.id,old.name,old.salary);

end

delete from emp where id=1005；//验证触发器

事务:需要搭配存储过程，需要让某些操作同时执行成功或者都不执行(封装)

事务的特性：一致性：事务完成之后，所有数据应保持一致

原子性：事务中的SQL语句要同时执行成功或者都不执行隔离性：事务之间互不影响

持久性：不可逆，事务一旦执行完毕，所有的数据改变在系统中永久有效

事务间的隔离级别：

Read-uncommited:读取未提交的数据，脏读Read-commited读取提交过的数据

Repeatable read可重复读Serializable序列化，访问时上锁，用完才能其他的才能用

注Set autocommit（值默认为1打开自动提交，值为0关闭自动提交）

回滚 rollback（撤销，恢复到事务开始之前的状态）保存点：savepoint p1；

Start transaction ---开始事务

Update emp set salary=1500 where id=1005;

savepoint p1；//保存点

Update emp set salary=2500 where id=1006;

//Rollback；这行代码之前若有错则将整个事务回滚

Rollback p1;//回滚到指定的保存点

Commit ----提交真正将所有的改变反映到数据库

性能优化

Mysql优化查询性能的优化包括库表结构、建立合理的索引、设计合理的查询。

库表结构包括如何设计表之间的关联、表字段的数据类型等

正确地创建和使用索引是实现高性能查询的基础

编写查询语句时，应尽可能选择合适的索引以避免单行查找，尽可能的使用原生顺序从而避免额外的排序操作，并尽可能使用索引覆盖查询。我们通过响应时间来对查询进行分析，找出消耗时间最长的查询或者给服务器带来压力最大的查询，然后检查查询的schema、SQL和索引结构，判断是否有查询扫描了太多的行，是否做了很多额外的排序或者使用了临时表，是否使用了随机I/O访问数据，或者太多回表查询哪些不在索引中的列的操作。

设计合理的查询

避免请求不需要的数据或扫描额外的记录(扫描行数和返回行数进行衡量)：如在查询时如果仅需要查询结果集中的前某些行，查询语句的最后加上limit。在进行多表关联查询时应尽量避免使用select \*

是否需要将一个复杂的查询分成多个简单的查询：对每个表进行一次单表查询，然后将结果在应用程序中进行关联(分解的好处：缓存的效率更高，查询分解后，执行单个查询可以减少锁竞争，查询本身效率也会有所提升，可以减少冗余记录的查询)

MySQL执行查询具体的操作过程：

一是MySQL客户端/服务器通信，二者之间通信协议是“半双工”的，即在某一时刻只能有一方在发送数据。在任何一个时刻MySQL连接都有一个状态，该状态表示MySQL当前的工作，通过SHOW FULL PROCESSLIST命令查询状态。其中状态有Sleep、Query、Locked、Analyzing and statistics、Coping to tmp table、Sorting result、Sending data。

二是查寻缓存。在解析一个查询语句之前，如果查询缓存是打开的，那么MySQL会优先检查这个查询是否命中查询缓存中的数据。通常是通过一个对大小写敏感的Hash查找实现。如果命中，那么在返回结果前MySQL会检查一次用户权限，该过程无须解析查询SQL语句。如果未命中，则解析SQL语句。

三是查询优化处理。包括解析SQL、预处理、优化SQL执行计划，其中出现任何错误都会终止查询。首先，MySQL通过关键字将SQL语句进行解析，并生成一棵对应的“解析树”。查询优化器负责将解析树转化成执行计划，优化器的作用就是找到查询的较优执行计划。MySQL使用基于成本的优化器，它将尝试预测一个查询使用某种执行计划时的成本（SHOW STATUS LIKE 'Last\_query\_cost'），并选择成本最小的一个。优化策略分为：静态优化和动态优化。静态优化可以直接对解析树进行分析，并完成优化。静态优化在第一次完成后就一直有效，即使使用不同的参数重复执行也不会发生变化，可以认为是一种“编译时优化”。动态优化是上下文相关的，如where条件中取值、索引条目对应的数据行数等，是一种“运行时优化”。

四是查询执行引擎。MySQL根据执行计划给出的指令逐步执行，在该过程中，有大量的操作需要通过调用存储引擎实现的接口来完成，也就是“Handler API”。MySQL在优化阶段就为每个表创建一个handler实例，优化器根据这些实例的接口获取表的相关信息。

五是将查询的结果返回给客户端。MySQL将结果集返回客户端是一个增量、逐步返回的过程。一旦服务器处理完最后一个关联表，开始生成第一条结果时，MySQL就可以开始想客户端逐步返回结果。两个好处：一是服务器端无须存储太多的结果；二是结果集中的每一行都会以一个满足MySQL客户端/服务器通信协议的封包发送，再通过TCP协议进行传输，从而是客户端可以在第一时间获得返回的结果。