mysql笔记

终端内启动 mysql -u root -p（回车后输入密码）

DDL

数据库操作：

1、查看已建立的数据库show databases;

2、新建数据库 create database name;

3、打开数据库 use name;

4、查看所选数据库 select database();

5、删除数据库 drop database name;

表操作：

1、展示该数据库内所有表show tables;

2、新建表 create table name(字段 字段类型 comment ‘备注’，字段 字段类型);

3、编辑已建立表 alter table name add/modify/change/drop/rename to; (add添加字段，modify更改已有字段的类型，change更改字段名称和类型，drop删除字段，rename to更改表名)

4、删除表drop table name;

5、查看表 desc 表名;

DML

对表中数据记录进行操作：

1、指定字段添加数据 insert into 表名(字段名1,字段名2) values(值1,值2);

2、给全部字段添加数据 insert into 表名 values(值1,值2...);

3、批量添加数据insert into 表名 (字段名1,字段名2。。。) values (值1,值2),(值1,值2),(值1,值2); //或 insert into 表名 values (值1,值2),(值1,值2),(值1,值2);

(值的顺序和字段一一对应，值在字段类型范围内，日期和字符串双引号)

4、修改数据 update 表名 set 字段名1 = 值1,字段名2 = 值2 where 条件 //where可写可不写，不写时全体批量修改

5、删除数据 delete from 表名 where 条件 //where同上

DQL单表查询

查询数据库表中的记录：

1. 基本查询

查询多个字段 select 字段1,字段2,字段3... from 表名;

select \* from 表名;

设置别名 select 字段1 as 别名1,字段2 as 别名2... from 表名;

去除重复记录 select distinct 字段列表 from 表名;

1. 条件查询（where）

语法 select 字段列表 from 表名 where 条件列表;

条件常用运算符 比较运算符：> >= < <= = <>或!=//不等于 between...and...

in(...)//在括号内的列表中满足其中一个值

like 占位符//模糊匹配（\_匹配单个字符，%匹配任意个字符）

is null

逻辑运算符：and或&& or或|| not或!

1. 聚合查询（count、max、min、avg、sum）

聚合函数-将一列数据作为整体纵向计算 count//统计数量 max//最大值 min//最小值 avg//平均值 sum//求和

语法 select 聚合函数(字段列表) from 表名;

1. 分组查询（group by）

语法 select 字段列表 from 表名 where 条件 group by 分组字段名 having 分组过滤后条件 //where在分组之前进行过滤，having对分组过后的结果进行过滤

1. 排序查询（order by）

语法 select 字段列表 from 表名 order by 字段 排序方式;

升序asc 降序desc

1. 分页查询（limit）有复合要求时limit排在所有条件之后

语法 select 字段列表 from 表名 limit 起始索引,每页查询记录数;

起始索引=（查询页码-1）\*每页显示的数量

DQL语句的编写顺序：select 字段列表 from 表名 where 条件列表 group by 分组字段列表 having 分组后条件列表 order by 排序字段列表 limit 分页参数；

DQL语句的执行顺序：from > where >group by > select > order by > limit //说明where无法调用select中给字段列表起的别名，只能使用固定名称

DCL

数据控制语言，管理数据库用户、控制访问权限：

1. 用户管理
2. 查询用户

use mysql；//打开数据库

select \* from user；//显示用户列表

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\_password by ‘密码’;

1. 删除用户

drop user ‘用户名’@‘主机名’;

//主机名可以使用%统配

1. 权限控制

|  |  |
| --- | --- |
| 权限 | 解释 |
| all ,all privileges | 所有权限 |
| select | 查询数据 |
| insert | 插入数据 |
| update | 修改数据 |
| alter | 修改表 |
| drop | 删除数据库/表/视图 |
| create | 创建数据库/表 |
| delete | 删除数据 |

1. 查询权限

show grants for ‘用户名’@‘主机名’；

1. 授予权限

grant 权限 on 数据库名.表名 to ‘用户名’@‘主机名’；

1. 撤销权限

revoke 权限 on 数据库名.表名 from ‘用户名’@‘主机名’；

//数据库名和表名均可使用\*进行通配，\*代表所有

函数select 函数名(参数)

1、字符串函数

|  |  |
| --- | --- |
| concat(S1,S2,...Sn) | 将括号内对象拼接成一个字符串 |
| lower(str) | 将字符串全部转为小写 |
| upper(str) | 将字符串全部转为大写 |
| lpad(str,n,pad) | 左填充，用字符串pad对str的左边进行填充，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，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 |

实际事例（填充表内字段）：update table set counter = lpad(counter,6, ‘0’);

2、数值函数

|  |  |
| --- | --- |
| ceil(x) | 向上取整 |
| floor(x) | 向下取整 |
| mod(x,y) | 取x模y |
| rand() | 返回0～1内的随机数 |
| round(x,y) | 求x四舍五入的值，保留y位小数 |

实际事例（生成一个六位数的随机验证码）：select lpad(round(rand()\*1000000,0),6,’0’);

1. 日期函数

|  |  |
| --- | --- |
| 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 |
| 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 type)  例select date\_add(now(),interval 70 year); | 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) | 返回起始时间date1和结束时间date2之间的天数 |

实际事例（按照在职时间倒序查询）：select name,datediff(curdate(),entrydate) as entrydays from emp order by entrydays desc;//entrydate、emp、name为事例数据库内表和字段

1. 流程函数

|  |  |
| --- | --- |
| if(value,t,f) | 如果value为true则返回t，否则为f |
| ifnull(value1,value2) | 如果value1不为空，返回value1，否则为value2 |
| case when [val1] then [res1] ... else [default] end | 如果val1为true，返回res1，...否则返回default莫认知 |
| 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 |

实际事例（统计班级成员成绩）：select case when (math>=80 then ‘优秀’ when math<60 then ‘不及格’ else ‘及格’ end)’数学’ from score;

约束

作用于表中字段上的规则，用于限制存储在表中的数据，保证数据的正确,可在创建/修改表时添加，多个约束之间仅需用空格分隔

|  |  |  |
| --- | --- | --- |
| 分类 | 描述 | 关键字 |
| 非空约束 | 该字段不可为null | not null |
| 唯一约束 | 该字段所有数据不重复 | unique |
| 主键约束 | 主键是一行数据的唯一表示，非空且唯一 | primary key |
| 默认约束 | 保存数据时若未指定该字段的值，则采用默认值 | default |
| 检查约束 | 保证字段值满足某一条件 | check |
| 外键约束 | 让两张表之间建立连接，保证数据的一致性和完整性 | foreign key |

外键约束：使两张表的数据之间建立连接，保证数据的一致性和完整性

具有外键的为子表，所连接的为父表

建立关联 语法 1）[constraint] [外键名称] foreign key (外键字段名) references 主表(主表列名) //在最初创建时设置 2）alter table 表名 add constraint 外键名称 foreign key (外键字段名) references 主表 (主表列名) //在后期修改时设置

删除/更新行为：

|  |  |
| --- | --- |
| no action | 默认，在父表中删除/更改时若有外键则不允许操作 |
| restrict | 默认，同上 |
| cascade | 在父表中删除/更改时，外键在子表中同步变化 |
| set null | 在父表中删除/更改时，外键在子表中更新为null |
| set default | 父表变更时，子表中外键变为默认值 |

语法 alter table add constraint 外键名称 foreign key (外键字段) references 主表名 (主表字段名) **on update cascade on delete cascade //在更新时的操作，删除时的操作**

多表查询

多表关系、内外连接、自连接、子查询

一对多：在多的一方建立外键，指向一的主键

多对多：建立第三张中间表，至少包含两个外键，分别关联双方的主键

一对一：在任意一方加入外键，关联另一方的主键

语法 sleect \* from 表1表2; //得到笛卡尔积，两表中对象一一匹配（输出数量为两表所有的组合情况）

消除无效笛卡尔积：设置条件为外键字段=主表对应字段 select \* from 表1表2 where 外键字段=主表对应字段;distinct去重

内连接：查询A、B两表交集部分数据

隐式内连接：select 字段列表 from 表1,表2 where 条件;（外键等于主表对应字段）

显式内连接：select 字段列表 from 表1 [inner] join 表2 on 连接条件;

//如果给表起了别名，就不能通过原表名设置限定字段

外连接：左外连接——查询左表所有数据及两表交集部分数据

select 字段列表 from 表1 left [outer] join 表2 on 条件;

查询表1的数据并包含两表并集部分

右外连接——查询右表所有数据及量表交集部分数据

select 字段列表 from 表1 right [outer] join 表2 on 条件;

查询表2的数据并包含两表并集部分

自连接：当前表与自身的连接查询，自连接必须使用表的别名

select 字段列表 from 表A 别名1 join 表A 别名2 on 条件;

//可为内连接也可为外连接

联合查询 union，union all

多次查询的结果合并，形成新的查询结果集

select 字段列表 from 表1;

union all; //若去重则只用union，union all为直接拼接结果

select 字段列表 from 表2;

//多张查询表的列数必须一致，字段类型也要一致

子查询；在sql语句中嵌套select语句

select \* from t1 where column1=(select column1 from t2);

外部语句可为insert、update、delete、select中的一个

|  |  |
| --- | --- |
| 标量子查询（=、<>、>、>=、<、<=） | 结果为单个值 |
| 列子查询（in、not in、any、some、all） | 结果为一列 |
| 行子查询（=、<>、in、not in） | 结果为一行 |
| 表子查询（in） | 结果为多行多列 |

//<>为不等于

列子查询：in-指定范围内多选一、not in-不在指定范围内、any-满足任意一个、some-等同于any、all-所有值必须满足 select \* from table where column in (select column1 from table2 where a=1 or a=2);

表子查询：select t1.\*,t2.\* from (select \* from table where a>b) t1 left join table2 t2 on t1.m=t2.n;

事务

简介：一组操作的集合，所有操作作为一个整体一起向系统提交/撤销操作，要么同时成功，要么同时失败。结构：开始事务-回滚事务-提交事务

基本操作

查看/设置事务提交方式 select @@autocommit;//自动提交

set @autocommit=0;//手动提交

提交事务 commit;//手动提交时在执行commit前其他命令都是临时改动

回滚事务 rollback;

开启事务 start transaction;或begin;

原子性：事务是最小操作单元

一致性：事务完成后所有数据保持一致

隔离性：数据库系统不受外部操作影响

持久性：提交/回滚对数据永久改变

并发事务问题：

|  |  |
| --- | --- |
| 脏读 | 一个事务读取到另一事务未提交的数据 |
| 不可重复读 | 一个事务先后读取同一条记录。但两次的数据不同 |
| 幻读 | 一个事务按条件查询时，无对应数据行，插入时数据又已存在 |

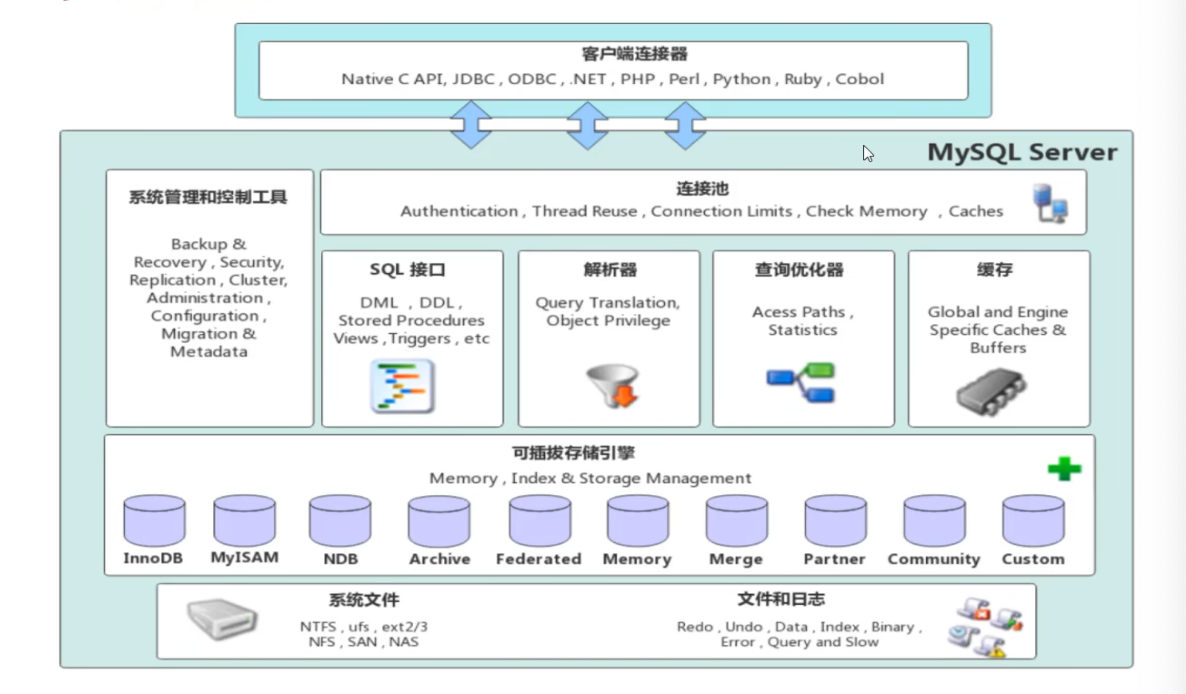
事务隔离级别：解决并发事务的问题

|  |  |  |  |
| --- | --- | --- | --- |
| 级别 | 脏读 | 不可重复读 | 幻读 |
| read uncommitted |  |  |  |
| read committde | 1 |  |  |
| repeatable read（默认） | 1 | 1 |  |
| serializable | 1 | 1 | 1 |

查看隔离级别select @@transaction\_isolation

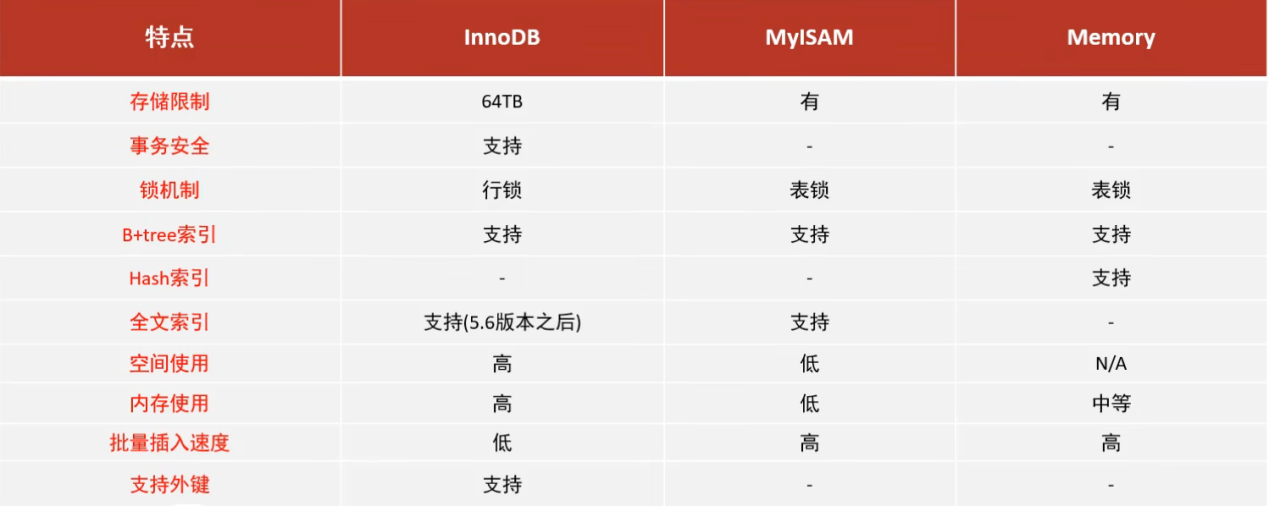
设置隔离级别set transaction isolation level read uncommitted;

MySQL体系结构 连接层、服务层、引擎层、存储层



1. 指定引擎：create table 表名(字段 字段类型)engine=innodb;
2. 查看支持引擎：show engines;
3. 查看系统变量：show variables like ‘innodb\_file\_per\_table’

|  |  |  |
| --- | --- | --- |
| innodb | 高可靠性高性能；  支持事务，DML遵循ACID模型；  行级锁，提高并发访问性能；  支持foreign key约束 | 每张表对应一个ibd表空间文件，包含表结构、数据和索引参数  innodb\_file\_per\_table |
| mylsam | 不支持事务、外键  支持表所，不支持行锁  访问速度快 | 文件：sdi表结构信息  myd数据  myi索引 |
| memory | 作临时表或缓存  内存存放  支持hash索引 | 文件：sdi表结构信息 |



索引

概述：index有序的的数据结构，引用/指向数据

无索引：查询方式为全表查询

有索引：如二叉树，会指向对应数据的存储地址，更高效

优点：提高数据检索的效率，降低数据库io成本

通过引列对数据进行排序，降低排序成本，降低cpu消耗

缺点：索引列也占用空间

降低了表的更新速度，需要同步更改数据结构

索引结构：在存储引擎层实现，不同的存储引擎结构也有所不同

|  |  |
| --- | --- |
| B+Tree索引 | 最常见，大部分引擎支持 |
| Hash索引 | 底层结构由哈希表实现，精确匹配才有效，不支持范围查询 |
| R-tree索引（空间索引） | MyISAM引擎的特殊索引，用于地理空间数据类型 |
| Full-text索引（全文索引） | 建立倒排索引，快速匹配文档 |

|  |  |  |  |
| --- | --- | --- | --- |
| 索引 | Innodb | MyISAM | Memory |
| B+tree | 支持 | 支持 | 支持 |
| Hash | 不支持 | 不支持 | 支持 |
| R-tree | 不支持 | 支持 | 不支持 |
| Full-text | 支持 | 支持 | 不支持 |

二叉树：当顺序插入时，会形成链表，查询性能降低，大数据量层次深，检索速度慢

红黑树：解决树的平衡问题，但大数据量下层级深检索速度慢

BTree多路平衡查找树：以最大度数为n的b-tree（每个节点最多存储n-1个key，n个指针）

B+Tree：最大度数为n的B+Tree所有的元素都会出现在叶子节点，非叶子节点主要起索引的作用，还增加了指向相邻叶子结点的链表指针，形成了带有顺序指针的B\_Tree，提高了区间访问的性能

Hash：采用一定的hash算法，将键值换算成新的hash值，映射到对应槽位上，存储在hash表中

特点：只能用对等比较（=，in），不支持范围查询（between，>，<，...）；无法利用索引完成排序操作；查询效率高，通常只需要一次检索，效率通常高于B+tree

InnoDB自适应hash索引

|  |  |  |  |
| --- | --- | --- | --- |
| 主键索引 | 对表中主键的索引 | 默认自动创建，只能有一个 | PRIMARY |
| 唯一索引 | 避免同一个表中某数据列中值重复 | 可有多个 | UNOIQUE |
| 常规索引 | 快速定位特定数据 | 可有多个 |  |
| 全文索引 | 查找文本中关键词 | 可有多个 | FULLTEXT |

InnoDB中存储形式分类索引：

聚集索引--将数据存储和索引在一块，索引结构的叶子节点保存行数据--必须有，且只有一个

二级索引--将数据与索引分开，索引结构的叶子结点关联对应的主键--可以存在多个

·创建索引：create [unique|fulltext] index index\_name on table\_name (index\_col\_name,...);

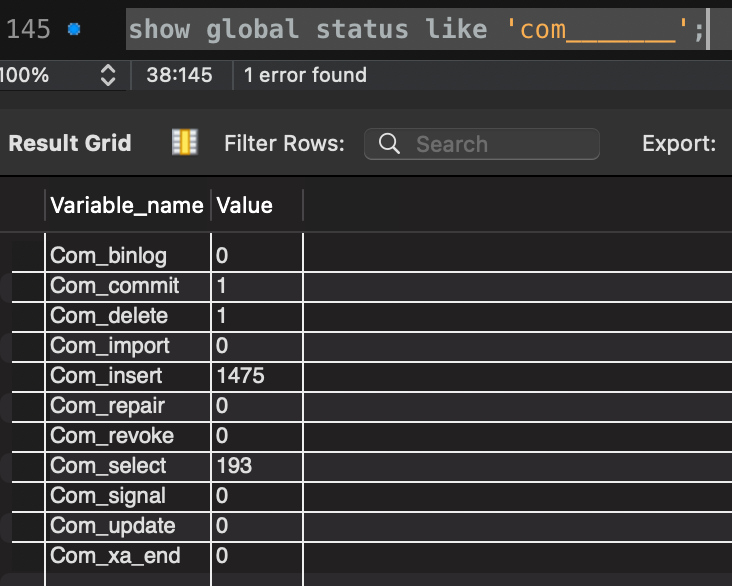
一个索引可关联多个字段，单个为单列索引，多个为联合索引/组合索引

·查看索引：show index from table\_name;

·删除索引：drop index index\_name on table\_name;

sql性能分析--主要优化查询语句

sql执行频率 show [session（当前会话） | global（全局）] status提供服务器状态信息。通过show global ststus like ‘com\_\_（模糊匹配，下划线数对应字符数）’;可查询数据库insert、update、delete、select的访问频次



慢查询日志：记录所有执行时间超过指定参数（long\_query\_time,默认10s）的所有sql语句show variables like ‘slow\_query\_long’;查询是否开启慢查询日志，在配置文件（/etc/my.cnf）中配置语句show\_query\_log=1//开启 long\_query\_time=2//设置时间为2s