# sql基础

**mysql语法规范和sql常识**

SQLyog中：

F12对选中的sql语句进行格式化

F9对选中的sql语句进行执行

不区分大小写,包括数据内容

分号结尾

单行注释 #

多行注释 /\*\*/

退出命令： exit;

count(\*)和count(1)效果一样，相当于加了一个常量列

+号会将字符运算转为数值运算，无法转换为数字则转为0，拼接字符串要使用concat()

**简称**

|  |  |
| --- | --- |
| DBMS（database Management system） | 数据库管理系统（mysql、oracle、sqlserver） |
| SQL （structure Query Language） | 结构化查询语言 |
| DDL （data define language） | 数据定义语言 |
| DQL （data query language） | 数据查询语言，select |
| DML （data manipulation language） | 数据操纵语言，insert update delete |

**数据类型**

**数值**

|  |  |  |
| --- | --- | --- |
| tinyint | 1 byte | 超小整数 |
| smallint | 2 byte | 小整数 |
| mediumint | 3 byte | 中整数 |
| int、integer | 4 byte | 整数 |
| biging | 8 byte | 大整数 |
| float | 4 byte | 单精度浮点数 |
| double | 8 byte | 双精度浮点数 |
| decimal(m,d) | m+2 byte | 定点数（m:整数位数+小数位数  d:小数位数） |

**字符**

|  |  |  |
| --- | --- | --- |
| char(n) | b byte | 固定长度 |
| varchar(n) | n+2 byte | 可变长度 |
| text |  |  |
| blob |  | 二进制 |

**日期**

|  |  |  |
| --- | --- | --- |
| date | 4 byte |  |
| datetime | 8 byte |  |
| timestamp | 4 byte |  |
| time | 3 byte |  |
| year | 1 byte |  |

**日期格式符:**

|  |  |
| --- | --- |
| %Y | 4位的年份 |
| %y | 2位的年份 |
| %m | 2位月份 |
| %c | 1/2位月份 |
| %d | 2位日期 |
| %H | 24小时制 的小时 |
| %h | 12小时制 的小时 |
| %i | 2位分钟 |
| %s | 2位秒 |

**约束**

|  |  |
| --- | --- |
| primary key | 主键:该字段不能有重复，也不能为null  一个表只能有一个主键，可以是复合主键 |
| default | 默认:字段有默认值 |
| not null | 非空:不能为空 |
| unique | 唯一:该字段不能有重复，可以为null允许复合唯一 |
| check（mysql不支持） | 检查约束：写条件限制字段内容 |
| foreign key  （所在表为从表） | 外键：该字段是主表的主键，且类型一致  该字段值必须是主表中主键字段的值 |

**标识列**

auto\_increment:自增标识列

**常用命令**

查看数据库： SHOW DATABASES;

使用数据库： **USE** 数据库名;

展示表： SHOW TABLES;

展示表结构： **DESC** 表名

**DDL**

**库管理**

创建： **DROP** **DATABASE** 表名**;**

删除： **CREATE** **DATABASE** 表名**;**

修改字符集：**ALTER** **DATABASE** 表名 **CHARACTER** **SET** 字符集**;**

使用数据库：**USE** 数据库名**;**

**表管理**

**创建**

**CREATE** **TABLE** 表名**(**

字段名 字段类型 **[**列级约束**],**

字段名 字段类型 **[**列级约束**],**

字段名 字段类型 **[**列级约束**],**

表级约束

**);**

列级约束：对外键无效，不能添加约束名

表级约束：不支持非空/默认，可以添加约束名，会有默认约束名

[constraint] 约束名 约束类型(字段名)

**删除**：**DROP** **DATABASE** staff**;**

**修改表结构**

查看表结构： **DESC** 表名**;** 或 SHOW COLUMNS **FROM** 表名**;**

改字段名： **ALTER** **TABLE** 表名 CHANGE **COLUMN** 字段名 新字段名 新字段名类型**;**

改字段类型/约束： **ALTER** **TABLE** 表名 **MODIFY** **COLUMN** 字段名 新字段类型**;**

添加字段： **ALTER** **TABLE** 表名 **ADD** **COLUMN** 字段名 字段类型**;**

删除字段： **ALTER** **TABLE** 表名 **DROP** **COLUMN** 字段名**;**

改表名： **ALTER** **TABLE** 表名 **RENAME** **TO** 新表名**;**

复制表结构： **CREATE** **TABLE** 表名 **LIKE** 被复制表名**;**

复制表结构和数据：**CREATE** **TABLE** 表名 **SELECT** **\*** **FROM** 被复制表名 **WHERE** 1**;**

查看表约束： SHOW **INDEX** **FROM** 表名**;**

添加列级约束：**ALTER** **TABLE** 表名 **MODIFY** **COLUMN** 字段名 类名 约束名**;**

添加表级约束：**ALTER** **TABLE** 表名 **[ADD** **CONSTRAINT** 约束名**]** 约束类型**(**字段名**) [外键的引用];**

删除列级约束：**ALTER** **TABLE** 表名 **MODIFY** **COLUMN** 字段名 类名**;**

删除表级约束：**ALTER** **TABLE** 表名 **DROP** 约束类型 约束名**;**

drop primary key

drop index 约束名

drop foreign key 约束名

**DQL**

**执行顺序（按照执行顺序，要设了别名才能用）**

from

join on

where

group by

having

select

order by

limit

**基础查询**（查询列表可以是字段、常量值、表达式、函数，查询结果是虚拟表）

**SELECT** 查询列表 **FROM** 表名**;**

**取别名：**使用 AS|空格 取别名，当别名包含特殊字符（空格）时，需要加引号

**SELECT** 字段名 别名 **FROM** 表名**;**

给表取了别名，则不能使用 原表名.字段名

**去重：DISTINCT**关键字，只允许一个字段

**SELECT** **DISTINCT** 字段名 **FROM** staff**;**

**条件查询**

**SELECT** 查询列表 **FROM** 表名 **WHERE** 筛选条件**;**

条件判断符：**>** 、**<** 、 **=** 、**!=** 、**<>** 、**>=** 、**<=**、**<=>**

逻辑判断符：**AND** 、**OR** 、**NOT** 、**&&** 、**||** 、**!**

模糊查询：**LIKE** 、**BETWEEN** **AND** 、**NOT** **BETWEEN** **AND**、**IN** 、**IS** **NULL**

可以使用 != && || ! 但不推荐，可读性查

= 只能判断普通内容

is 只能判断null

<=> 能判断普通内容和null，很少用，可读性差

**分页查询**

**SELECT** 查询列表 **FROM** 表名 **LIMIT** [起始索引]**,数据个数;**

省略起始索引，则从0开始

分页查询公式：limit (page-1)\*size size

**常见函数**

**SELECT** 函数名(实参列表) [**FROM** 表名]

sql中索引从1开始，函数可以嵌套调用

单行函数：**concat()**、**IFNULL()**等

字符函数：**LENGTH()、CONCAT()、CONCAT()、SUBSTR()、SUBSTR()、INSTR()、TRIM()、LPAD()、REPLACE()**

数学函数：**SELECT** **ROUND()**、**SELECT** **CEIL()**、**SELECT** **FLOOR()**、**SELECT** **TRUNCATE()**、**SELECT** **MOD()**、**SELECT** **RAND()**

日期函数：NOW**()**、CURDATE**()**、CURTIME**()**、year**()**、**month()**、day**()**、**hour()**、**minute()**、**second()** STR\_TO\_DATE**()**、**DATE\_FORMAT()**、**DATEDIFF()**

其他函数：VERSION**()**、**DATABASE()**、**USER()**

流程控制函数：**IF CASE**

分组函数（统计函数，聚合函数）：**sum()**、**avg()**、**max()**、**min()**、**count()**

1.sum和avg只处理数值，max,min,count可以处理任何类型

2.分组函数都忽略Null值

3.可以个distinct搭配去重

4.count(\*)/count(1)统计行数

5.和分组函数一同查询的字段 必须是group by的字段

**分组查询**

**SELECT** 分组函数，字段（**GROUP** **BY**分组字段）

**FROM** 表名

**WHERE** 分组前筛选

**GROUP** **BY** 分组的列表

**HAVING** 分组后筛选（分组函数做条件判断）

**ORDER** **BY** 排序

**多表连接查询**

**sql99语法**

**SELECT** 字段

**FROM** 表1

**INNER|LEFT|RIGHT** **JOIN** 表2 **ON** 连接条件

**内连接**（inner join/join）:等值连接、非等值连接、自链接

**外连接**=内连接结果+主表中有但从表中没有的记录

从表中无则，字段显示为null

左外连接(left [outer] join)：左边的是主表

右外连接(right [outer] join)：右边的是主表,很少用右外连接，可以转换

全连接(full [outer] join)：左外和右外的并集

交叉连接(cross join) :左外和右外的交集

**子查询：**

子查询需要加()

每个派生表都必须有自己的别名

子查询结果：

标量子查询：一行一列,一般搭配着>、<、>=、<=、=、<>

列子查询： 一列多行，一般搭配 in/not in、any/some(任一)、all(所有)

行子查询： 一行多列

表子查询： 多行多列

分类

select后面： 标量子查询

from后面： 表子查询

where/having后面：标量子查询，列子查询，行子查询

exists后面（子查询有数据为1，无数据为0）： 表子查询

**联合查询**

union ：将多条查询语句的结果合并成一张表（类似or，可读性高）

列名和列数要一致，默认自动去重，union all不去重

**DML**

**插入语句:insert**

**INSERT** **INTO** 表名**(**字段名1**,**字段名2**,...)** **VALUES(**字段值**,** 字段值**,...);**

**修改:update**

单表修改： **UPDATE** 表名 **SET** 字段**值=**字段名 **WHERE** 条件**;**

多表连接修改：**UPDATE** 表名1 **INNER|LEFT|RIGHT** **JOIN** 表名2 **ON** 连接条件 **SET** 字段**值=**字段名 **WHERE** 条件**;**

**删除:delete**

清空整张表： **TRUNCATE** **TABLE** 表名**;**

单表删除记录： **DELETE** **FROM** 表名 **WHERE** 条件**;**

多表连接删除纪录：**DELETE** 表名1 **FROM** 表名1 **INNER|LEFT|RIGHT** **JOIN** 表名2 **ON** 连接条件 **WHERE** 条件**;**

delete删除数据后，再插入数据，自增列列原基础上自增, 支持回滚

truncate删除后，自增列会从1开始，不支持回滚

**视图：是一张虚拟表，可以普通表一样使用**

**作用：**sql语句较复杂且使用频繁，不妨将该查询封装（只保存sql逻辑）到视图中（一张虚拟的表），拿出来就能用

表数据改变，对应的视图也会改变，对视图进行增删改，对应表数据也会改变（一般不对视图做改动）

可以在视图上建视图

**使用：**就当一张表来使用

**创建：**只保存sql逻辑，表数据改变，对应的视图也会改变，可以在视图上建视图

**CREATE** **VIEW** 视图名 **AS** **SELECT** 查询语句**;**

**修改：**

方式1：**CREATE** **OR** **REPLACE** **VIEW** 视图名 **AS** **SELECT** 查询语句**;**

方式2：**ALTER** **VIEW** 视图名 **AS** **SELECT** 查询语句**;**

**删除： DROP** **VIEW** 视图名1**,**视图名2**;**

**查看视图结构： DESC** 视图名**; 或** SHOW **CREATE** **VIEW** 视图名**;**

**以下视图不允许更新**

包含分组函数、distinct、group by、having、union、union all

常量视图

select中包含子查询

join

from 一个不能更新的视图

where子句的子查询引用了from子句中的表

**级联删除的外键**

alter table 表名 add constraint 约束名 foreign key(字段名) references 主表名(字段名) on delete caseade;

**级联制空**

alter table 表名 add constraint 约束名 foreign key(字段名) references 主表名(字段名) on delete set null;

**事务**

**视图特性：**原子性、一致性、隔离性、持久性

**隐式事务**:insert,update,delete

**显示事务：**多条语句，需要自行开启

**SET** autocommit **=** 0**;**开**启**事务：

**START** **TRANSACTION;** 可写也可不写

**SQL**语句；

**SQL**语句；

**COMMIT;** 提交事务

**ROLLBACK;** 回滚事务

**SAVEPOINT** a**;** 设置保存点

**ROLLBACK** **TO** a**;** 回滚到保存点

**事务隔离级别**

**会出现的问题**

脏读： 读到其他事务更新但没提交的数据，之后该数据又被回滚，读到了其他事务**更新**的数据

不可重复读：两次读取之间，有其他事务对数据进行了更新

幻读： 读表中数据，两次读取的行数不同，因为读到了其他事务**插入**的数据

**隔离级别（级别越高，越安全，性能越低）**

|  |  |  |
| --- | --- | --- |
| **READ UNCOMMITTED** | 读未提交数据 | 不可避免问题 |
| **READ COMMITED** | 读已提交数据 | 避免脏读，只读已提交的事务 |
| **REPEATABLE READ** | 可重复读 | 避免脏读和不可重复读，禁止更新 |
| **SERIALIZABLE** | 串行化 | 避免所有问题，禁止其他事务进行插入、更新、删除操作 |

**设置隔离级别**

查看当前隔离级**别： SELECT** @@tx\_isolation**;**

设置当前mysql连接隔离级**别：SET** **TRANSACTION** **ISOLATION** **LEVEL** **READ** COMMITTED**;**

设置全局隔离级**别： SET** **GLOBAL** **TRANSACTION** **ISOLATION** **LEVEL** **READ** COMMITTED**;**

**变量**

系统变量

全局变量

会话变量

自定义变量

用户变量

局部变量

查看系统变量: show [global] variables;

查看满足条件的系统变量： show [global] variables like ‘%char%’ //对变量名进行模糊查询

查看指定的某个系统变量： selelct @@[global.]系统变量名;

设置系统变量：

set [global] 系统变量名=值

set @@[global.]系统变量名 = 值

默认都是session,添加 global为全局变量

用户变量初始化/赋值： set @用户变量名:=值;

局部变量: 只能在begin end块中

声明： declare 变量名 类型 default 值;

赋值： set局部变量名:=值

**函数和存储过程**

sql函数和存储过程部分

批量插入数据

# 索引调优

**索引**

**索引(index)**：高效获取排好序数据并快速查找的数据结构，B树索引。

类似于图书馆的书目，可以快速查找的图书

优点：对数据排序建立索引，提高检索效率缺点：

缺点：索引也要占空间，会降低更新表的速度，不仅要更新数据，还要跟新对应的索引信息。

**索引分类：**

单值索引：该索引只包含单个字段

唯一索引：索引列的值必须唯一，单允许有空值，就是unique

复合索引：该索引包含多个字段

名词-覆盖索引：只用查询字段为建索引的字段

**索引管理**

索引创建1：create [unique] index 索引名 on 表名(字段名1,字段名2;

索引创建2: alter 表名 add [unique] index 索引名 on 字段名;

索引删除:drop index 索引名 on 表名;

索引查看:show index from 表名;

**索引结构**

BTree索引

Hash索引

full-text全文索引

1. Tree索引

**需要建索引的情况**

主键自动建立唯一索引

查询条件中频繁出现的字段，要建立索引

外键关系，要建立索引

单值/组合索引选择：高并发下倾向于组合索引

查询需要排序的字段建立索引，会提高索引排序速度，因为索引就已经排序了

查询中统计/分组的字段,要建立索引

左连接 索引加右边，因为左表一定全都有，会全表扫描  
 右链接 索引加左边，同理

**不需要建索引的情况**

频繁更新的字段，不建立索引

where条件中用不到的，不创建索引

表数据太少

经常增删改的表

数据列包含许多重复的内容，建立索引就没有太大的效果

不对范围查询的字段建索引

**索引失效情况**

  最佳左前缀法则，如果使用复合索引，索引左边字段不能丢失，中间字段不能跳过，顺序可以不同，sql中会按索引字段顺序重新排序  
  不要在索引列上做任何操作（计算，函数 ，隐式or手动类型转换，隐式的转换为等号左边的类型）  
  范围条件之后的索引都会失效，范围之后索引全失效  
  尽量使用覆盖索引，（索引列和查询列一致），减少select\*  
  不使用!=、<>、is null、is not null   
  like以通配符开头会变成全表扫描"%asf"，可以用覆盖索引解决%aa% 两边%索引失效  
  用or连接会索引失效  
  字符串不加单引号 索引会失效，varchar类型绝对不能失去单引号，是重罪，会隐式类型转换为字符串，类型转换会使索引失效

**索引优化**  
 **join语句优化**  
  尽量减少join语句中nestedloop的循环总次数，用小结果集驱动大结果集  
  优先优化nestedloop的内层循环  
  保证join语句被驱动表的join条件字段被索引  
 无法保证被驱动表的join条件字段被索引且内存资源充足时，不要吝啬joinBuffer的设置（缓存）

**索引建议**

对于单值索引，尽量选择针对当前query过滤性更好的索引

对于复合索引，当前query中过滤性最好的字段放在索引靠前的位置，尽量选择能够包含当前query中的where字段中更多字段的索引

尽可能通过分析统计信息和调整query的写法来达到选择合适索引的目的

永远小表驱动大表：条件既能放where也能放having，那么优先放where，(小表驱动大表)

select \* from A where id in (select id from B)

等价于(B表数据小于A表时适用,in)

for selelct id from B

for selelct \* from A where A.id = B.id

select \* from A where exists (select 1 from B where B.id=A.id)

等价于（B表数据大于A表时适用,exists,替代 in）

for selelct \* from A

for selelt \* from B where B.id=A.id

order by子句的排序优化

避免使用selelct \*

尽量使用index方式排序，避免使用fileSort方式排序

尽量再索引列上完成排序操作，遵循最佳左前缀

如果进行filesort，会有单路排序/双路排序

增大sort\_buffer\_size参数，

增大 max\_length\_for\_sort\_data参数，提高排序效率

group by子句的优化

先排序后分组，会产生临时表

order by中的优化都能用在group by中

where 高于having,能在where中限定的条件就不要去写having

**explain插件：查看索引使用情况**

用法：explain sql语句

id字段：id越大越先执行，id相同时，由上往下执行

**select\_type：查询类型**

simple:简单select查询，不包含子查询/union

primary: 复杂子查询的最外层

subquery：select/where中包含子查询

deriued：from列表中包含的子查询

union: 第二个select出现union后，表记为union,若Union包含在from子句中的子查询中，外层select表记为derived

union result：从union表中获取结果的select

**type：索引级别**

从好到差： system>const>eq\_ref>ref>range>index>all

一般查询要达到range级别，最好达到ref

system:表中只有一行记录，系统表

const:通过索引一次就能找到，用于primary key/unique索引，只匹配一行数据

eq\_ref：唯一性索引扫描，表中只匹配一条数据，用于primary key/unique索引

ref：非唯一索引扫描，返回匹配某个值的所有行，匹配到多行，是查找和扫描的混合体

range：值检索给定范围的行， where语句中出现between,<,>,in等，比全表扫描好一点

index：从索引遍历全表，比all块

all：从硬盘遍历全表

**possible\_keys:查询可能用到的索引**

**key: 查询实际用到的索引，查询中使用了覆盖索引，则指挥出现在key列表中**

**key\_len: 索引中使用的字节数**

**ref: 显示索引的哪一列被使用了，哪些列/常量被用于查找索引列上的值**

**row:有多少行被优化器查询到， 越少越好**

**extra：额外的信息**

using filesort(使用文件内排序):不按索引排好的顺序读取,重新排序(九死一生)

using temporary(使用临时表):order by /group by中使用临时表保存中间结果，(拖慢sql的元凶，要少使用)

using index(使用索引):select使用了覆盖索引，效率不错，

如果同时using where，表明索引被用来执行索引键值的查找

如果没有，则表明索引读取数据而非执行查找

using where:使用where过滤

using join buffer:使用join缓存

impossible where: where子句总是false,不能用来获取任何数据

selelct tables optimized away:没有group子句的情况下，在min/max/count(\*)在查询执行计划生成时完成，不在执行计划阶段再进行计算

distince: 在找到第一匹配的数据(元组)后，停止找同样是数据

**mysqldumpslow插件**

慢查询

记录执行时间超过(大于)阈值的语句，默认 long\_query\_time为10秒

默认没有开启慢查询日志，需要手动设置慢日志参数，一般只在调优的时候用

show variables like ‘%slow\_query\_log%’ 查看参数

set global show\_query\_log=1; 开启慢查询日志

mysqldumpslow插件帮助分析慢查询语句的信息

show profile

mysql提供可以用来分析当前会话中语句执行的资源消耗情况，用于sql调优的测量

默认关闭，并保存最近15次的运行结果

使用

查看mysql版本是否支持 show variables like ‘profiling’;

开启功能 set profiling=on;

运行sql

查看结果 show profiles;

诊断SQL, show profile cpu,block io for query 上一步前面的问题SQL数字号码；

日常开发需要注意的结论

converting HEAP to MyISAM 查询结果太大，内存都不够用了，往磁盘上搬

creating tmp table 创建临时表，拷贝数据到临时表，用完再删除

copying to tmp table on disk 把内存中临时表复制到磁盘，危险

locked

全局查询日志：只能再测试环境中用

**锁机制**

mysql锁机制

操作类型划分

读锁（共享锁）：多个操作同时进行，不会相互影响

阻塞其他会话的写

写锁（排他锁）： 当前操作完成前，其他操作会阻塞

阻塞其他会话的读写

搜索引擎：InnoDB与MyISAM

innoDB:

支持事务，支持行锁,支持行锁，不仅缓存索引，还缓存真实数据，

MyIsAM：支持表锁，只缓存索引，不缓存真实数据，

颗粒度划分

表锁(偏读，锁表)

偏向MyISAM存储引擎，开销小，加锁块，无死锁，锁定粒度大，发生锁冲突概率高，并发度低

行锁（锁一行）：

偏向InnoDB存储引擎，开销大，加锁慢，有可能死锁，锁定粒度最小，发生冲突概率低，并发度高

mysql是可重复读的事务级别，避免了脏读和不可重复读

查看当前事务隔离级别 show variables like ‘tx\_isolation’

优化建议

尽可能让所有数据检索都通过索引完成，无索引使行锁升为表锁

尽量缩小锁的范围

尽量较少检索条件，避免间隙锁

控制事务大小，减少所锁定资源量和时间长度

尽可能低级别事务隔离

间隙锁

索引失效，行锁会变表锁

查看锁：show open tables;

锁类型

read: 读锁

write：写锁

添加锁：lock table 表名 锁类型，表名 锁类型;

释放锁：unlock tables;

添加read锁

|  |  |
| --- | --- |
| session1 | session2 |
| 创建连接，表A获得read锁 | 创建连接 |
| session1可以查询表A记录，不能查询没有锁的表 | 可以查询任何表 |
| session1插入/更新锁定表数据会报错 | session2插入/更新锁定表会 阻塞，直到获得锁 |
| 释放表A的read锁 | session2获得锁，更新操作执行并完成 |

添加write锁

|  |  |
| --- | --- |
| session1 | session2 |
| 创建连接，表A获得read锁 | 创建连接 |
| 可对锁定表进行 查询，更新，插入 | 查询、更新等操作都会阻塞 |
| 释放锁 | 操作执行并完成 |