# MySQL

# 01 简介

数据库分类：（1）文档型如sqllite （2）服务型如mysql，终端以tcp/ip协议连接，进行数据读写工作

E-R模型（**E-entry实体；R-relationship关系**）

**E对应数据库中一个表**；**R描述两个实体之间的对应规则**，包括一对一、一对多、多对多。R转换为数据库表中一个字段（列）

**三范式**

1. 列不可拆分
2. 唯一标识（通过一个字段唯一确定一个对象（行））
3. 引用主键

**数据类型**

1. 数字:

int（整数）

decimal（小数，指明一共多少位以及小数点后多少位）：decimal(5,3)

1. 字符串：

char（固定字符，不足右侧用空格不足）：char(8)

varchar（可变字符）：varchar(8)

text（不能预知字符串长度）

1. 日期：datetime
2. 布尔：bit，默认1位，bit(4)表示4位

**约束**

1. 主键约束：primary key 唯一
2. 非空：not null
3. 唯一：unique
4. 默认：default
5. 外键：foreign key

逻辑删除：

对于重要数据，需要增加isDelete字段，0表示存在，1表示逻辑删除，但实际未删除

# 02 命令脚本操作

（1）连接数据库

* 本地连接：mysql -uroot -p：u后加用户名 p回车后输入密码
* 远程连接：mysql -hip地址 -P 端口 -uroot -p

（2）退出

* quit或exit

（3）查看版本及时间（命令后注意**分号**）

* 查看版本：select version();
* 显示当前时间：select now();

（4）数据库操作

* 新建数据库：create database db\_name;
* 显示所有数据库：show databases;
* 删除数据库：drop database db\_name;
* 切换数据库：use db\_name;
* 查看当前选择的数据库：select database();

（5）表操作

* 新建表：

create table person(

id int **auto\_increment primary key** not null,

name varchar(10) not null,

gender bit default 1,

birthday datetime);

create table name(

field1 type auto\_increment primary key not null(约束),

field2 type (约束),

field3 type (约束))

* 查看所有表：show tables;
* 删除表：drop table table\_name;
* 修改表：

增加列：alter table table\_name **add** column\_name type constraints

删除列：alter table table\_name **drop** **column** column\_name

修改列：

修改字段属性：alter table table\_name **change** column\_name new\_type

new\_constraints

修改字段名称和属性：alter table table\_name **modify** column\_name new\_column\_name new\_type new\_constraints

将test字段改为test1

alter table 表名 change 原字段名 新字段名 字段类型 约束条件

alter table user10 change test test1 char(32) not null default '123';

* 查看表结构：desc 表名;
* 更改表名称：rename table 原表名 to 新表名;
* 查看表的创建语句：show create table 表名;

（5）数据操作

* 整表查询：select \* from 表名
* **增加记录**：

全列插入：insert into 表名 values(d1,d2,d3),(b1,b2,b3);

values()中顺序与表中字段对应，values后一个括号表示一条记录

缺省插入：insert into 表名(c1,c2) values(d1,d2,d3),(b1,b2,b3);

表名后的括号表示部分字段，values()中数据与其一一对应。

* 记录修改：update 表名 set 列1=值1，列2=值2 where 条件
* 物理删除：delete from table where 条件;
* 逻辑删除：alter table students add isdelete bit default 0;

update students isdelete=1 where 条件;

（6）数据库备份与恢复

* 数据库备份：

sudo s

cd /var/lib/mysql

mysqldump –uroot –p 数据库名 > ~/Desktop/备份文件.sql;

* 数据库恢复

新建空数据库

mysql -uroot –p 数据库名 < ~/Desktop/备份文件.sql

# 03 条件查询

1. **select**

select \* from 表名; 查询全部字段

select 列1,列2 from 表名; 查询其中两列  
select **distinct**列1 from 表名; 消除重复数据

1. **where**条件（对记录进行筛选）

select \* from students where ID >= 3;

逻辑运算符：and or not(非)

* where 列名 not 条件语句;
* select \* from students where id <3 and not gender=0

1. **like 模糊查询**

%表示任意多个任意字符

\_表示一个任意字符

select \* from students where **name like** 'ws\_';

1. **范围查询**

in(1,2,8) 不连续范围

between 3 and 8 连续范围

select \* from students where ID between 3 and 5;

1. **空判断**

is null

is not null

select \* from students where birthday is not null;

1. **优先级**

**小括号，not，比较运算符，逻辑运算符**

and比or先运算，如果同时出现并希望先算or，需要结合()使用

# 04 聚合

**将现有多行数据进行统计合并**

count(\*) 统计总行数 select count(\*) from students;

max(列名) 求某一列最大值 select max(birthday) from students;

min(列名) 求某一列最小值 select min(birthday) from students;

sum(列名) 求某一列的和 select sum(ID) from students where gender=1;

avg(列名) 求某一列的平均值 select avg(ID) from students where gender=1;

# 05 分组

按照字段（1个或多个）分组，字段相同的数据会被放到一个**组**中。分组后，**只能查询出相同的数据列**，对于有差异的数据列无法出现在结果集中，**可以对分组后的数据进行统计**，做聚合运算（分组的目的是为了聚合）。

1. 分组关键字group by

select gender as 性别,count(\*) from students group by gender; **统计各性别的记录总数，**as 性别是为分组字段取别名

1. 分组筛选关键字having

select gender as 性别,count(\*) from students group by gender having gender is not null; **在性别非空条件下，统计各性别的记录总数, having 后的字段在分组字段范围内或对聚合结果进行筛选**

1. having与where的区别

where是对from后面指定的表进行数据筛选，属于对**原始数据的筛选**

having是对group by的**结果**进行**筛选**

select gender as 性别,count(\*) as rs from students group by gender having rs > 2;

按照性别分组，然后求每组的记录总数，并筛选记录总数大于2的结果

# 06 排序

关键词：order by：**位置在where之后**

select \* from 表名

order by 列1 asc|desc,列2 asc|desc,...

* 将行数据按照列1进行排序，如果某些行列1的值相同时，则按照列2排序，以此类推
* 默认按照列值**从小到大**排列
* **asc从小到大排列，即升序**
* **desc从大到小排序，即降序**

# 07 分页

减轻数据库或用户的压力

语法：

**select \* from 表名limit start, count**

**start: 开始数据索引（从0开始）**

**count：取得数据长度**

# 08 关键字顺序

（1）完整语句

**select distinct \***

**from 表名**

**where ....**

**group by ... having ...**

**order by ...**

**limit star,count**

（2）执行顺序

**from 表名**

**where ....**

**group by ...**

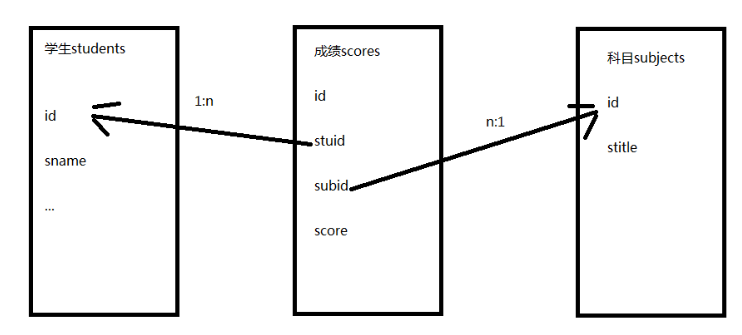
**select distinct \***

**having ...**

**order by ...**

**limit star, count**

# 09 关系



1对多：在多方侧引用外键

多对多：建立中间表

**关系不能闭合**

**创建表时添加外键：**

create table scores(

id int primary key auto\_increment,

stuid int,

subid int,

score decimal(5,2),

**foreign key(stuid) references students(id),**

**foreign key(subid) references subjects(id)**

);

**创建表之后添加外键：**

alter table scores

**add constraint** scores\_students\_ID\_ID\_fk

foreign key (stuid) references students (ID);

**外键的级联操作**

* 在删除students表的数据时，如果这个id值在scores中已经存在，则会抛**异常**
* 推荐使用逻辑删除解决这个问题
* 可以**创建表时指定级联操作**，也可以在**创建表后再修改外键的级联操作：**

alter table scores add constraint stu\_sco foreign key(stuid) references students(id) on delete **cascade**;

**级联操作的类型包括：**

* restrict（限制）：默认值，抛异常
* **cascade（级联）：如果主表的记录删掉，则从表中相关联的记录都将被删除**
* set null：将外键设置为空
* no action：什么都不做

# 10 连接查询

示例：查询每个学生每个科目的分数

select students.sname,subjects.stitle,scores.score

from scores

**inner join** students on scores.stuid=students.id

inner join subjects on scores.subid=subjects.id;

连接查询的分类：

* 表A inner join 表B：表A与表B匹配的行会出现在结果中（**交集**）
* 表A left join 表B：表A与表B匹配的行会出现在结果中，外加表A中独有的数据，未对应的数据使用null填充（**以左表为准**）
* 表A right join 表B：表A与表B匹配的行会出现在结果中，外加表B中独有的数据，未对应的数据使用null填充（**以右表为准**）

注意事项：

* 在查询或条件中推荐使用“**表名.列名**”的语法
* 如果多个表中列名不重复可以省略“表名.”部分
* 如果表的名称太长，可以在表名后面使用**' as 简写名'或' 简写名**'，为表起个临时的简写名称

# 11 自关联查询

场景：省、市、区信息建一张表

意义：存储的都是地区信息，而且每种信息的数据量有限，没必要增加一个新表，或者将来还要存储区、乡镇信息，都增加新表的开销太大。

解决方案：

定义表areas，结构如下

* id（主键）
* title
* pid（设置外键为id，省记录为null，市记录指向省id，区记录指向市id）

这张表既可以当省用，也可以当市或县区用，从而实现自关联查询，如查询山东所有的市。

（1）查询省的名称为“山东省”的所有城市（一级连接）

select city.\* from areas as city //市表

inner join areas as province on city.pid = province.id //与省表连接查询

where province.title = "山东省"; //条件

（2）查询省的名称为“山东省”的所有城市及区县（二级连接）

select city.\*,county.\* from areas as city

inner join areas as province on province.id = city.pid //市表连接省表

left join areas as county on city.id = county.pid //县表连接市表

where province.title = "山东省";

# 12 视图

本质：SQL查询语句的一个封装

语法：

create view **stuscore** as

xxx(SQL查询语句)

使用：select \* from stuscore;

# 13 事务

当一个业务逻辑**需要多个sql**完成时，如果其中**某条sql语句出错，则希望整个操作都退回**。

使用事务可以完成**退回**的功能，保证业务逻辑的正确性。

表的类型必须是**innodb或bdb**类型，才可以对此表使用事务

alter table '表名' engine=innodb; 修改表的数据引擎

**事务四大特性(简称ACID)**

* 原子性(Atomicity)：事务中的全部操作在数据库中是**不可分割**的，要么全部完成，要么均不执行
* 一致性(Consistency)：几个并行执行的事务，其**执行结果必须与按某一顺序串行执行的结果相一致**
* 隔离性(Isolation)：事务的执行不受其他事务的干扰，**事务执行的中间结果对其他事务必须是透明的（加锁）**
* 持久性(Durability)：对于任意已提交事务，**系统必须保证该事务对数据库的改变不被丢失**，即使数据库出现故障

事务语句：

开启begin;

提交commit;

回滚rollback;

使用事务的场景：

数据更改：insert、update、delete、alter

# 14索引

索引作用：提高数据访问性能

**主键**和**唯一索引**都可以作为索引

选择索引的原则：

1. 数据类型越小、越简单越好（如整数型就好于字符串）
2. 避免索引存在NULL（可以用0代替）

索引分类：

单列索引：一个索引包含单列

多列索引：一个所用包含多列

操作

查看索引：show index from 表名

创建索引：create index 索引名 on 表名(字段名(长度))

删除索引：drop index [索引名] on 表名

缺点：

提高查询速度，但会降低表更新速度，如insert、update、delete，因为更新表时，不仅需要保存数据，还需要保存索引文件（占用磁盘空间）

时间监测：

set profiling = 1

show profiles

# 15内置函数

**（1）字符串函数**

* 查看字符的ascii码值ascii(str)，str是空串时返回0：**select ascii('a')**;
* 查看ascii码值对应的字符char(数字)：**select char(97)**;
* 拼接字符串concat(str1,str2...)：**select concat(12,34,'ab')**;
* 包含字符个数length(str)：**select length('abc')**;
* 截取字符串：
* **left(str,len)** 返回字符串str的左端len个字符
* **right(str,len)** 返回字符串str的右端len个字符
* **substring(str,pos,len)** 返回字符串str的位置pos起len个字符
* 去除空格：
* **ltrim(str)** 返回删除了左空格的字符串str
* **rtrim(str)** 返回删除了右空格的字符串str
* **trim([方向 remstr from str)** 返回从某侧删除remstr后的字符串str，方向词包括both、leading、trailing，表示两侧、左、右
* 返回由n个空格字符组成的一个字符串：**select space(n)**
* 替换字符串：**select replace(str,from\_str,to\_str)**
* 大小写转换：**lower(str) upper(str)**

**（2）数学函数**

* 绝对值：abs
* 取余：mod(%)
* floor：向下取整
* ceiling：向上取整
* round(n,d)：四舍五入保留指定小数位数
* 求幂：pow(2,3)
* 圆周率：PI()
* 0-1随机数：rand()

**（3）日期时间函数**

* 获取子值
* year(date) 返回date的年份(范围在1000到9999)
* month(date) 返回date中的月份数值
* day(date) 返回date中的日期数值
* hour(time) 返回time的小时数(范围是0到23)
* minute(time) 返回time的分钟数(范围是0到59)
* second(time) 返回time的秒数(范围是0到59)
* 日期计算，使用+-运算符，数字后面的关键字为year、month、day、hour、minute、second：**select '2016-12-21'+interval 1 day**;
* 日期格式化：date\_format(date, format)，format参数可用的值如下：
* **获取年%Y，返回4位的整数**
* **获取年%y，返回2位的整数**
* 获取月%m，值为1-12的整数
* 获取日%d，返回整数
* **获取时%H，值为0-23的整数**
* **获取时%h，值为1-12的整数**
* 获取分%i，值为0-59的整数
* 获取秒%s，值为0-59的整数

select date\_format('2016-12-21','%Y %m %d');

* 获取当前时间及日期
* 当前日期：current\_date()
* 当前时间：current\_time()
* 当前日期时间：now()

# Redis

# 01 简介

NoSQL，全名为Not Only SQL，指的是非关系型的数据库

优点:

* 高可扩展性
* 分布式计算
* 低成本
* 架构的灵活性，半结构化数据
* 没有复杂的关系

缺点:

* 没有标准化
* 有限的查询功能（到目前为止）
* 最终一致是不直观的程序

非关系型数据库分类：

|  |  |  |
| --- | --- | --- |
| 类型 | 部分代表 | 特点 |
| 列存储 | Hbase  Cassandra  Hypertable | 顾名思义，是按列存储数据的。最大的特点是**方便存储结构化和半结构化数据**，方便做数据压缩，对针对某一列或者某几列的查询有非常大的IO优势。 |
| 文档存储 | **MongoDB**  CouchDB | 文档存储一般用类似json的格式存储，存储的内容是文档型的。这样也就有有机会对某些字段建立索引，实现关系数据库的某些功能。 |
| key-value存储 | Tokyo Cabinet / Tyrant  Berkeley DB  MemcacheDB  **Redis** | 可以**通过key快速查询到其value**。一般来说，存储不管value的格式，照单全收。（Redis包含了其他功能） |
| 图存储 | Neo4J  FlockDB | 图形关系的最佳存储。使用传统关系数据库来解决的话性能低下，而且设计使用不方便。 |
| 对象存储 | db4o  Versant | 通过类似面向对象语言的语法操作数据库，通过对象的方式存取数据。 |
| xml数据库 | Berkeley DB XML  BaseX | 高效的存储XML数据，并支持XML的内部查询语法，比如XQuery,Xpath。 |

# 02 运行与配置

* 启动服务器：redis-server（按ctrl+c停止）
* 启动客户端：redis-cli
* 修改配置文件：sudo vim /etc/redis/redis.conf
* 配置文件方式启动：sudo redis-server /etc/redis/redis.conf
* 停止redis服务：ps ajx|grep redis

sudo kill -9 redis的进程id

# 02 数据操作

redis是**key-value**的数据，所以每个数据都是一个**键值对**

* 键的类型是字符串
* 值的类型分为五种：
* 字符串string
* 哈希hash
* 列表list
* 集合set
* 有序集合zset

进入redis：

redis-cli

命令查询网站：

<http://redis.cn/commands.html>

**（1）字符串**

string是redis最基本的类型

最大能存储**512MB**数据

string类型是二进制安全的，即可以为**任何数据，比如数字、图片、序列化**对象等

基本命令：

* 设置
* 设置键值**：set key value**
* 设置键值及过期时间，以秒为单位：**SETEX** key **seconds** value
* 设置多个键值：**MSET** key value [key value ...]
* 获取
* 根据键获取值，如果不存在此键则返回nil：GET key
* 根据多个键获取多个值MGET key [key ...]
* 运算（要求值数数字）
* 将key对应的value加1：INCR key
* 将key对应的value加整数：INCRBY key 10
* 将key对应的value减1：DECR key
* 将key对应的value减整数：DECRBY key 10
* 其它
* 追加值：APPEND key value（在value后追加）
* 获取值长度：STRLEN key

**（2）键操作**

* 查找键，参数支持正则：**KEYS pattern**
* 判断键是否存在，如果存在返回1，不存在返回0：**EXISTS key [key ...]**
* 查看键对应的value的类型：**TYPE key**
* 删除键及对应的值：**DEL key [key ...]**
* 设置过期时间，以秒为单位：**EXPIRE key seconds**
* 查看有效时间，以秒为单位：**TTL key**

**（3）Hash（对象：json格式，由键值（field-value）对组成）**

* 设置
* 设置单个属性：**HSET key field value**
* 设置多个属性：**HMSET key field value [field value ...]**
* 获取
* 获取一个属性的值：**HGET key field**
* 获取多个属性的值：**HMGET key field [field ...]**
* 获取所有属性和值：**HGETALL key**
* 获取所有的属性：**HKEYS key**
* 返回包含属性的个数：**HLEN key**
* 获取所有值：**HVALS key**
* 其它
* 判断属性是否存在：**HEXISTS key field**
* 删除属性及值：**HDEL key field [field ...]**
* 返回值的字符串长度：**HSTRLEN key field**

**（4）List（数组，元素类型为String）**

* 设置
* 在头部插入数据：LPUSH key value [value ...]
* 在尾部插入数据：RPUSH key value [value ...]
* 在一个元素的前|后插入新元素：LINSERT key BEFORE|AFTER pivot value
* 设置指定索引的元素值，索引是基于0的下标索引可以是负数，表示偏移量是从list尾部开始计数，如-1表示列表的最后一个元素：LSET key index value
* 获取
* 移除并且返回 key 对应的 list 的第一个元素：LPOP key
* 移除并返回存于 key 的 list 的最后一个元素：RPOP key
* 返回存储在 key 的列表里指定范围内的元素，start 和 end 偏移量都是基于0的下标，偏移量也可以是负数，表示偏移量是从list尾部开始计数，如-1表示列表的最后一个元素：LRANGE key start stop
* 其它
* 裁剪列表，改为原集合的一个子集，start 和 end 偏移量都是基于0的下标，偏移量也可以是负数，表示偏移量是从list尾部开始计数，如-1表示列表的最后一个元素：LTRIM key start stop
* 返回存储在 key 里的list的长度：LLEN key
* 返回列表里索引对应的元素：LINDEX key index

**（5）set（无序集合，元素类型为String，不能重复）**

* 设置
* 添加元素：SADD key member [member ...]
* 获取
* 返回key集合所有的元素：SMEMBERS key
* 返回集合元素个数：SCARD key
* 其它
* 求多个集合的交集：SINTER key [key ...]
* 求某集合与其它集合的差集：SDIFF key [key ...]
* 求多个集合的合集：SUNION key [key ...]
* 判断元素是否在集合中：SISMEMBER key member

**（6）zset（有序集合，元素类型为String，不能重复）**

* 设置
* 添加：ZADD key score member [score member ...]（score为顺序）
* 获取
* 返回指定范围内的元素：ZRANGE key start stop
* 返回元素个数：ZCARD key
* 返回有序集key中，score值在min和max之间的成员：ZCOUNT key min max
* 返回有序集key中，成员member的score值：ZSCORE key member

# 03 发布订阅

**（1）消息格式**

推送消息的格式包含三部分

part1:消息类型，包含三种类型

* subscribe，表示订阅成功
* unsubscribe，表示取消订阅成功
* message，表示其它终端发布消息

如果第一部分的值为subscribe，则**第二部分是频道**，第三部分是**现在订阅的频道的数量**

如果第一部分的值为unsubscribe，则第二部分是频道，第三部分是现在订阅的频道的数量，如果为0则表示当前没有订阅任何频道，当在Pub/Sub以外状态，客户端可以发出任何redis命令

如果第一部分的值为message，则**第二部分是来源频道的名称**，**第三部分是消息的内容**

**（2）命令**

* 订阅：SUBSCRIBE 频道名称 [频道名称 ...]
* 取消订阅，如果不写参数，表示取消所有订阅：UNSUBSCRIBE 频道名称 [频道名称 ...]
* 发布：PUBLISH 频道 消息

# 04 主从配置

一个master可以拥有多个slave，一个slave又可以拥有多个slave，如此下去，形成了强大的多级服务器集群架构

比如，将ip为192.168.1.10的机器作为主服务器，将ip为192.168.1.11的机器作为从服务器

* 设置主服务器的配置：bind 192.168.1.10
* 设置从服务器的配置，在slaveof后面写主机ip，再写端口，而且端口必须写：
* bind 192.168.1.11
* slaveof 192.168.1.10 6379
* 在master和slave分别执行info命令，查看输出信息
* 在master上写数据：set hello world
* 在slave上读数据：get hello