## 数据类型######################################

* 整数

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 大小（byte） | 无符号范围 | 有符号范围 |
| tinyint | 1 | 0~255 | -128 ~ 127 |
| smallint | 2 | 0~65535 | -32768 ~ 32767 |
| mediumint | 3 | 0~16777215 | -8388608 ~ 8388607 |
| int | 4 | 0~4294967295 | -2147483648 ~ 2147483647 |
| integer | 4 | 0~4294967295 | -2147483648 ~ 2147483647 |
| bigint | 8 | 0  ~  18446744073709551615 | -9223372036854775808  ~  9223372036854775807 |

int和integer，反正用起来没区别。还有就是，如果不特别指定，默认都是带符号的数字。如果用整数做主键，最好声明成无符号的。

* 浮点数与定点数

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 字节 | unsigned | signed |
| float | 4 | 0,  (1.175494351E-38  ~  3.402823466E+38) | (-3.402823466E+38  ~  1.175494351E-38),  0,  (1.175494351E-38  ~  3.402823466E+38) |
| double | 8 | 0,  (2.2250738585072014E-308  ~  1.7976931348623157E-308) | (-1.7976931348623157E+308  ~  2.2250738585072014E-308),  0,  (2.2250738585072014E-308  ~  1.7976931348623157E+308) |
| decimal(M,D) | M+2 | 和double一样 | 和double一样 |

E代表科学计数法，E-38就是10^-38

其次，把0隔出来，是因为中间有一段没法表示，看着很蛋疼，但如果要求不是很高的话，这个范围其实够了。小么小的要死，大么大到离谱。

关于decimal(M，D)，M 表示数据的总长度(不包括小数点)，D 表示小数位。

例如：decimal(5,2) 123.45

这么一看还真是大方，7个字节。。。。为了点精度耗费这么大空间，但从使用角度看，耗费再多也值得。

* 时间

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 字节数 | 取值范围及格式 | 零值 |
| year | 1 | 1910~2155 | 0000 |
| date | 4 | 1000-01-01~9999-12-31 | 0000:00:00 |
| time | 3 | -838:59:59~838:59:59 | 00:00:00 |
| datetime | 8 | 1000-01-01 00:00:00  ~  9999-12-31 23:59:59 | 0000-00-00 00:00:00 |
| timestamp | 4 | 19700101080001  ~  20380119111407 | 00000000000000 |

timestamp是时间戳的意思。和DATETIME一样，可以包含日期和时间，但范围只有1970-01-01 00:00:01 ~ 2038，好处是只占4字节。另外这玩意不同长度代表的含义不一样。

| TIMESTAMP(14) | YYYY-MM-DD HH：MM：SS

| TIMESTAMP(12) | YYYY-MM-DD HH：MM：SS

| TIMESTAMP(10) | YYYY-MM-DD HH：MM

| TIMESTAMP(8)　| YYYY-MM-DD

| TIMESTAMP(6)　| YYYY-MM-DD

| TIMESTAMP(4)　| YYYY-MM

| TIMESTAMP(2)　| YY

* 字符串

char 定长字符串，长度自己定，别像傻逼一样乱定

varchar 变长字符串，长度自己定，别像傻逼一样乱定

tinytext 255byte

text 65535byte

mediumtext 16MB

longtext 4GB

enum（枚举） 只能取一个元素

set（集合） 取多个元素

// enum与set，不会有人用的。。。。

* 二进制

tinyblob

blob

mediumblob

longblob

范围和text一样。存存图片，视频，但一般也不会有人用，太浪费性能，除非是机密，需要重点保护的图片或是视频什么的。

## 常见操作###############################

show databases;

create database 数据库名；

drop database 数据库名；

（not）between and，这东西管用，但我见的不多，因为只有有范围的字段才能应用，比如数字。

% 通配符

\_ 单个字符

is （not）null

**group by，**根据什么来分组。

group by 属性名 （having 一般是函数）（with rollup）

group by单独使用是毫无意义的。

配合 group\_concat(列名) 函数，可以把一个组中指定某列的数据拼成一个字符串。然后代码用split取，高效的一逼。

*select group\_concat(name) from av\_actress group by company*

配合聚合函数用，主要是count（），sum（）这种玩意

*select count(\*) from av\_actress group by company*

配合rollup

*select group\_concat(name) from av\_actress group by company with rollup*

在最底下新增一行，这一行会把查询出来的数据直接累加。

**Limit**

// 这又是一个把人搞得糊里糊涂的玩意。。。。

select \* from t\_person limit 1；

select \* from t\_person limit 0，1；

select \* from t\_person limit 1 offset 0；

Limit的作用是返回规定的记录数，呐，我为什么说它烦呢，你往下看：

limit 1 这个意味着只取1条数据，从第0条开始取。

limit 0，1 这个意味着只取1条数据，从第0条开始取。

limit 1 offset 0 这个意味着只取1条数据，从第0条开始取。

limit 1，-1 这个意味着从第1条取到最后一条

........

我还能说什么。这个第0条也是个很恶心的玩意，有的地方从1开始，有的地方从0开始，我现在终于明白为什么标准这么重要了。

**聚合函数**

以下函数均带参数

count() // 统计列数，注意，列数，不是列中数据的总和

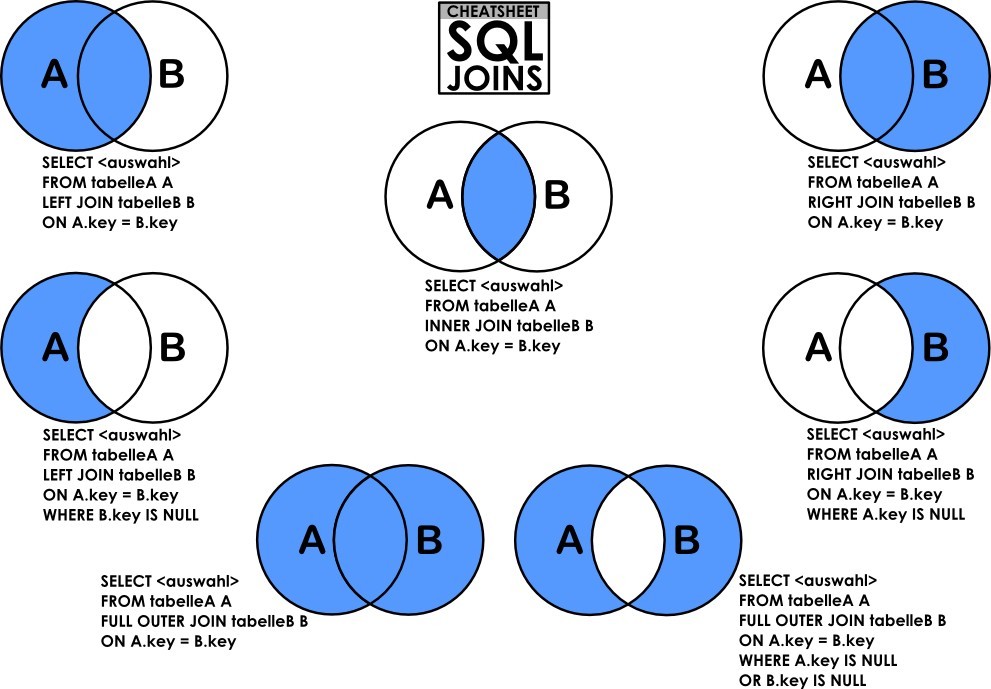
sum() // 统计总数

avg() // 平均值

max() // 最大值

min() // 最小值

**连接查询**



上图，没有什么比图片来的更实在的了。

笛卡尔积，就是枚举所有可能的关系。两个集合X和Y的笛卡尓积（Cartesian product），又称直积，表示为X × Y，第一个对象是X的成员而第二个对象是Y的所有可能有序对的其中一个成员。

* 内连接：

SELECT … FROM

join\_table [INNER] JOIN join\_table2

[ON join\_condition]

WHERE where\_definition

　//这是一个完整的语句，中括号内的东西可以不写。这里要注意的点，ON 与 WHERE。连接条件与查询条件，这是不一样的。连接条件一般就是主外键相等，查询条件是针对具体的数据。这也是外连接存在的原因。

* 外连接

外连接分为左、右、全，语法结构参考内连接。之所以管他叫外连接，是因为查询时 “不仅列出连接条件相匹配的行，还列出与符合WHERE过滤条件的数据行”

* 交叉连接

SELECT … FROM

join\_table1 CROSS JOIN join\_table2;

返回的是笛卡尔积，没有ON 和 WHERE。

* 自连接

自己连自己，就是用别名。

**子查询**

in，<，>，<>，（not）exists，any，all

这些关键字不一定非得用在子查询上，我只是顺带着提一提。另外exists还没用过，这玩意是用来判断的。假如子查询查到记录，则进行外层查询，未查到不进行。

select \* from av\_actress where exists (select \* from av\_company)

我也不是特别理解，用到了再说吧。

any关键字表示满足任一条件。

select \* from av\_actress where bust = any (1)

all关键字表示满足全部条件。

select \* from av\_actress where bust <= all (select bust from av\_actress)

这个sql可以找出奶子最小的女优。

不太会举例，就这样吧。in等价于= any，not in等价于<> all。

**union、union all**

合并结果集，用起来规矩还不少。

select \* from table1

union

select \* from table2

像上面那个例子非常有误导性，好像随便来两张表就可以直接用 union 合并结果集。假设第一个select语句的结果集为a，第二个select语句的结果集为b。那么，a的列数要与b的列数一致，同时，按照顺序，a中列的数据类型要与b中列的数据类型相似（数字对数字，）。

假设合并后最终结果集为c，那么c的列名是以a为准的。

union会去除重复项，union all保留重复项。

我目前还没有用过这两个东西，业务用不到啊。

insert into 表名（） values（）

update 表名 set 列名=

delete from 表名

now() 获得当前日期（包括时间在内） YYYY-MM-DD HH：mm：SS

sysdate() 效果和now()一样，只不过now()是在执行开始时就得到了值，而sysdate() 是在执行时动态的得到值。//完全不懂你在说什么。。。。

current\_timestamp()

group\_concat()

## 触发器###############################

这个东西还是挺好用的。一旦检测到某张表发生了变化，就自动执行一些SQL语句。比如a表与a\_log表，一旦a表插入了1条数据，那么就往a\_log表里插一条数据；如果a表删除了1条数据，那么也往a\_log表里插1条数据。

省得在代码里插数据，也因此一旦用了触发器，就不要在用代码实现一遍了，那不是脑子不好么。

**触发事件**有3中，INSERT、UPDATE、DELETE

**过度变量**，old，new

old与new代表具体的数据，而不是表。

old指刚刚被删除的数据

new指新加入的数据

**创建用SQL**

1. 执行1条SQL的触发器

CREATE TRIGGER 触发器名 AFTER 触发事件

ON 表名 FOR EACH ROW

要执行的SQL

；

1. 执行多条SQL的触发器

DELIMITER @

CREATE TRIGGER 触发器名 AFTER 触发事件

ON 表名 FOR EACH ROW

BEGIN

要执行的SQL 1;

要执行的SQL 2;

END

@

DELIMITER ；

// 触发器名，我的命名逻辑是“表名\_触发事件\_trigger”

**删除触发器** DROP TRIGGER 触发器名 ;

**显示触发器** SHOW TRIGGERS ;

// 这SQL你查一下就明白

select \* from information\_schema.triggers；

## 存储过程与函数###############################

省得老用代码写。

**创建语法**

CREATE PROCEURE 存储过程名 （[输入输出参数]）

////////////////////////////////////////////////////////////////////////////////////////////

## ~索引与视图

　　索引就是用来提高查询速度的，类似于目录，原理百度去。索引分6类。

* 普通索引
* 唯一性索引 UNIQUE // 该列不允许有重复值
* 全文索引 FULLTEXT // 这个索引只能创建在char varchar 还有 text 上，作用 是提高字符串的查询速度，像文章之类的，全文索引对中文 不友好。另外，全文索引只有MyISAM引擎支持，InnoDB 不支持（哈，总算知道MyISAM有什么用了，又不支持外 键，也不支持事务的家伙）
* 单列索引 // 这个内容和上面有点重复，他的意思就是只在一个列（字 段）上创建索引，和具体类型无关。
* 多列索引 // 多个列呗~ 光这么讲其实很无语，学习最令人头痛的事不 是原理不懂，而是不知道学的这玩意能干什么。MySQL查 询时只会使用一个索引，当查询的内容牵扯到多个索引时， 它会选里面限制最严格的。而多列索引牵扯的数据更多，因 此怎么看都比单列索引严格。

但是多列索引的使用是有条件的。假设有一个people表内 有firstName列与lastName列。然后给他们分别创建一 个单列索引。

再创建一个lastName，firstName的多列索引，这个顺序 很重要。

//这些是能用多列索引的

　　SELECT \* FROM test WHERE last\_name='Widenius';

　　SELECT \* FROM test WHERE last\_name='Widenius' AND first\_name='Michael';

　　SELECT \* FROM test WHERE last\_name='Widenius' AND (first\_name='Michael' OR first\_name='Monty');

　　SELECT \* FROM test WHERE last\_name='Widenius' AND first\_name >='M' AND first\_name < 'N';

//这个就不能用了

　　SELECT \* FROM test WHERE first\_name='Michael';

　　SELECT \* FROM test WHERE last\_name='Widenius' OR first\_name='Michael';

　　发现了吗，一是查找内容必须符合多列索引，而且得是AND运算符，再不济顺序要按照多列索引来。OR运算符与单列（不按照多列索引的顺序）就用不到。

嘿嘿，那多创点索引是不是查询效率就无敌了。naive！维护索引需要的开销是很大的。

* 空间索引 SPATIAL // 给空间数据提供，那么问题来了，空间数据是什么？

只有MyISAM支持。

////////////////////////////////////////////////////////////////////////////////////////////

## ~主外键关联

首先，主键的作用是唯一标识一条记录，不能有重复，不允许为空，用来保证数据的完整性。

而外键，是另一表的主键，外键可以有重复，可以是空值，用来和其他表建立联系。外键可以设多个，但也别激动的瞎设，不然没意义啊。

来两个例子，部门表的主键是员工表的外键，商家表的主键是食物表的外键。

、、、、、、、、、、、、、、、、、、、、、、、、、、使用外键的条件（妈的还有条件）

1. 两张表都是InnoDB表，且没有临时表

2. 建立外键关系的对应列必须具有相似的InnoDB内部数据类型。

3. 建立外键关系的对应列必须建立索引

4. 假如显式的给出了CONSTRAINT symbol，那symbol在数据库中必须是唯一的。假如没有显式的给出，InnoDB会自动的创建。

这段内容是我copy来的，基本不懂，但他提到了存储引擎的概念。mysql常见引擎有两种：InnoDB和MyISAM，后者不支持外键关系，也不支持事务。

、、、、、、、、、、、、、、、、、、、、、、、、、、创建语法

[CONSTRAINT symbol] FOREIGN KEY

[id] (从表的字段1)

REFERENCES

tbl\_name (主表的字段2)

[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]

[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION} ；

//这一长串是同一行，为了好看隔了开来，中括号的内容是可有可无的。

· constraint symbol 是指给外键约束起名字，不写的话，系统会默认创建。

· on delete 与 on update 是MySQL后来添加的参数，指操作主表记录时，所做的约定。

restrict（限制）：如果要删除的主表记录有对应的从表记录，那么不能删除或更新

cascade（级联）：如果主表记录删除了，相应的从表记录也会删除或更新

set null ：从表置空

no action：什么都不做

、、、、、、、、、、、、、、、、、、、、、、、、、、子查询与连接

　　比较意外的是，主外键的功能只在查询，挺。。那什么出乎我意料的。纠结了半天，还是把连接的语法重写一遍吧。

首先连接，你看名字就知道它必定涉及到两张表，因此，结果也必定涉及两张表。因为，如果你只想要其中一张表的内容，你跟本不需要连接语法啊。（自连接除外）

它其实是给表取两个别名，虚拟出两张表，然后来连接。换句话说，这个表内部必须有两列数据要类似主外键。唉，我暂时还没用到，用到了再说吧，语法是和内连接一样的。

////////////////////////////////////////////////////////////////////////////////////////////

/\*

\*\* 给建成的表添加属性，DDL，alter应用于多个场景，所以添加table来标识本次操作的\*\* 对象

\*/

alter table table\_name

add column\_name column\_type;

***增***

Insert into 表名 (属性名，...) values (具体值, ...)；

// 全填的话属性名可以不写

***Delete***

delete from table\_name where col\_name = ?；

上面的是删除一条数据，在where后使用in可以实现批量删除。

例： delete from t\_user where id in（1,2,3）；

还没完，MySQL可以多表删除，语法有两种

1. DELETE [LOW\_PRIORITY] [QUICK] [IGNORE]

tbl\_name[.\*] [, tbl\_name[.\*]] ...

FROM table\_references

[WHERE where\_condition]

2. DELETE [LOW\_PRIORITY] [QUICK] [IGNORE]

FROM tbl\_name[.\*] [, tbl\_name[.\*]] ...

USING table\_references

[WHERE where\_condition]

这意味着delete语句也可以使用连接作为删除条件，好屌啊。

***改***

update 表名 set 属性名=具体值，属性名=具体值，... ;

// 你们就不能统一格式吗！

***group by & having***

// 大小写没什么好讲究的，反正都一样。

这两个东西，暴力理解的话，需要函数查询语句里就用group by & having，不需要函数就正常用where。

group by是分组的意思，就是剔除列中的重复值，按它分组。Having是特殊的where，配合函数使用。长见识，原来where不能用函数啊。

***count（）***

count，统计，因此这个函数可以用来统计表内的数据数量。而且count()内可以使用distinct关键字，不统计重复的值。

select count(name) from person ;

上面这条语句会返回person表中name列的数量。但我奇怪的是这个：

select name from person having count(name) > 1;

这条语句就变成了查询person表中name数量不止一个的name。

这是为什么呢？因为一个在前一个在后？这么一看好像还真有点道理，select后的name代表“列”而having后的name代表“具体的数据”。