MYSQL

一、基本数据类型

在 SQL 语言 中，每个列、局部变量、表达式和参数都有一个相关的数据类型，这是指定对象可持有的数据类型（整型、字符、money 等等）的特性。下面列出系统提供的数据类型集：**精确数字，近似数字，字符串，Unicode 字符串，二进制字符串，其它数据类型。**

二、表、列、行

1、行：SQL 表中构成表中的水平行的元素集合。表中的每一行代表由该表建模的对象的一个出现，并存储该对象所有特性的值。

例如，在 Northwind 示例数据库中，Employees 表建立 Northwind Traders 公司职员的模型。表中的第一行记录职员 ID 为 1 的职员的全部信息（例如姓名和头衔）。

2、列：SQL 表中每一行中的区域，存储用该表制作模型的某个对象特性的数据值。

例如，Northwind 示例数据库中的 Employees 表建立 orthwind Traders 公司职员的模型。Employees 表的每个行中的 LastName 列存储该行所代表的职员的姓氏，就象窗口或表单中的 Last Name 字段包含姓氏一样。

三、查询

1、操作符

(1)算术操作符(+)

select discounttype,stor_id,lowqty,highqty,discount from discounts

现在我们把所有的折扣增加 0.5 元，可以键入 ：

select discounttype,stor_id,lowqty, highqty,**discount+0.5** from discounts

我们得到了想要的结果。

观察一下 discount+0.5 字段的标题是(无列名)，太不好听了，我们把它改一改。键入：

select discounttype,stor_id,lowqty, highqty,**discount+0.5 新折扣** from discounts

重新命名列标题。重新命名任意字段列标题的语法格式：列名 别名（注意它们之间有空格）

SQL 允许你将已有的列进行组合或计算，以**建立虚拟列**和导出字段，原始表并不发生变化。

（建立虚拟列）我们想把原来的折扣和新折扣加以对照，键入：

select discounttype,stor_id,lowqty,highqty,**discount 旧折扣**,**discount+0.5 新折扣** from discounts

算术操作符(-)

减号也有两种用法：可以用某列减去一个常数，或者用一列减去另一列。另一种用法是改变某数的符号。

用某列减去一个常数：

select discounttype,stor_id,lowqty, highqty, **discount-3** from discounts

用一列减去另一列，（省去了 as）键入：

select discounttype,stor_id,lowqty, highqty, **discount 旧折扣， discount+4 新折扣，discount+4-discount 折扣差** from discounts

算术操作符(/) 算术操作符(*) 算术操作符(%)

用某列除以/乘一个常数，或者用一列除以/乘另一列

模运算：

```
select discounttype,stor_id,lowqty, lowqty%3 模, highqty, discount  from discounts
```

(2)比较运算符
比较操作符比较两个表达式并返回如下三个值之一，**TRUE、FALSE 或 NULL**。
不能将空值用于区分表中两行所需的信息（例如，外键或主键）。
如果数据出现空值，则逻辑运算符和比较运算符有可能返回 TRUE 或 FALSE 以外的第三种结果 UNKNOWN。需要三值逻辑是导致许多应用程序出错之源。

比较运算符(=)
```
select * from discounts where discount = 5
```
上面字段 discount 的数据类型是数字型，对字符串类型的字段记得加上单引号：
```
select * from discounts where discounttype = 'Volume Discount'
```

比较运算符(>和>=)
```
select * from discounts where discount > 5
```
如果要包括 5：
```
select * from discounts where discount >= 5
```
字符也可以比较：
```
select * from discounts where discounttype >= 'Vo'
```

比较运算符(<和<=)
```
select * from discounts where discount < 6.7
```
如果要包括 6.7：
```
select * from discounts where discount <= 6.7
```
字符也可以比较：
```
select * from discounts where discounttype <= 'Vo'
```

比较操作符(**<>或!=**不等于)
```
select * from discounts where discount <> 6.7
```
还可以这样写：
```
select * from discounts where discount != 6.7
```
字符也可以不等于：
```
select * from discounts where discounttype <> ' Volume Discount '
```

(3)NULL
如果某记录的某字段里没有数据，则该字段的值就是 NULL。
NULL 并不意味着字段包含一个 0 值或长度为 0 的字符串。0 值是一个整数，而长度为 0 的字符串也是字符串的一个具体值。NULL 意味着什么也没有！

操作符 is null 用来检测变量是否为空值。
找出一列中值为 NULL 的记录：
```
select * from discounts where lowqty is null
```
如果用等号代替 is null：
```
select * from discounts where lowqty =  null
```

无输出，因为 lowqty = null 的比较结果是 FALSE。

还有一个操作符是 **is not null**

(4)字符操作符(like 和%)

如果你想查找不十分精确的数据，like 很好用：

select * from authors where au_lname like 'S%'

上面的操作选出 au_lname 的第一个字母是 S 的记录，试试小写

select * from authors where au_lname like 's%'

大小写没关系。

%是通配符，代表多个字符。%也可以多个使用，见下面例子。

%：表示任意数量的字符（包括 0 个字符）。它可以匹配一个或多个字符，

_：表示任意单个字符。它仅匹配一个字符的位置。

字符操作符(_)

下划线是单个字符通配符：

select * from authors where zip like '946_9'

多个下划线使用：

select * from authors where zip like '9_6_8'

_与%混合使用：

select * from authors where phone like '%9_6_8%'

字符操作符(+)

连接符(+)用于连接两个字符串：

select au_id, au_lname, au_fname, au_lname+au_fname from authors

用 concat，as 用于起别名，重新创建一个新的列名（可以省略）：SELECT au_id, au_lname, au_fname, CONCAT(au_lname, au_fname) AS full_name FROM authors;

(5)逻辑运算符(and) 逻辑运算符(or) 逻辑运算符(not)

select * from discounts where discount>=5 and discount<10

select * from discounts where discount=5 or discount=6.7

select * from discounts where not discount = 5

(6)操作符(in)

in 可以简化你已经学过的一些查询。我们看前面的一个例子：

select * from discounts where discount=5 or discount=6.7

用 in 来实现。

select * from discounts where discount in (5,6.7)

语句更短了且更容易阅读，in 也可用于字符类型的字段。

操作符(between)

between 也用来简化你已经学过的一些查询。我们看前面的一个例子：

select * from discounts where discount>=5 and discount<10

用 between 来实现。

select * from discounts where discount between 5 and 10

语句更短了且更容易阅读，between 也可用于字符类型的字段。
2.函数
(1)聚集函数(count)
count 函数返回符合 select 语句中的查询条件的行数。
select count(*) from discounts where discount>=5 and discount<10

聚集函数(sum)
sum 函数返回一列中所有值的和。
select sum(discount) from discounts where discount>=5 and discount<10
只对数值类型字段有效。

聚集函数(avg)
avg 函数计算一列的平均值。
select avg(discount) from discounts where discount>=5 and discount<10
只对数值类型字段有效。

聚集函数(max/min)
max 函数找到一列的最大值。
select max(discount) from discounts where discount>=5 and discount<10
可用于字符型字段。
min 函数找到一列的最小值。
select min(discount) from discounts where discount>=5 and discount<10
可用于字符型字段。
max 函数和 min 函数可以一块儿使用，查出值所在的范围。
select min(discount), max(discount) from discounts

聚集函数(var)
var 函数计算方差。
select var(discount) from discounts
只对数值类型字段有效。

聚集函数(stdev)
stdev 函数计算标准差。
select stdev(discount) from discounts
只对数值类型字段有效。

(2)日期函数(年、月、日)
给定日期取年、月、日
写法一：SELECT "Year Number" = **YEAR**('03/12/2003'),"Month Number" = **MONTH**('03/12/2003'), **DAY**('03/12/2003') AS 'Day Number'
写法二：SELECT "Year Number" = **DATEPART**(yy, '03/12/2003'), "Month Number" = **DATEPART**(mm, '03/12/2003'), "Day Number" = **DATEPART**(dd, '03/12/2003')
注意不同的列标题的定义方法，比较一下我们以前学过的别名。

日期函数(系统日期)
从系统日期中取年、月、日。
SELECT "Year Number" = YEAR(getdate()),"Month Number" = MONTH(getdate()),DAY(getdate())
AS 'Day Number'

日期函数(访问字段)
从日期字段中取年、月、日。
SELECT "Year Number" = YEAR(hire_date),"Month Number" = MONTH(hire_date),DAY(hire_date)
AS 'Day Number' from employee

(3)字符函数(chr)
将 int ASCII 代码转换为字符的字符串函数，例如：
SELECT char(65)
整数表达式是介于 0 和 255 之间的整数。如果整数表达式不在此范围内，将返回 NULL 值。
对于 Unicode 编码，整数表达式可以大于 255，例如：
SELECT nchar(65)
SELECT nchar(26578)

字符函数(len)
len 函数返回给定字符串表达式的字符（而不是字节）个数，其中不包含尾随空格。
SELECT discount, len(discount) from discounts
len 函数可用于数字和字符，例如：
SELECT * , len(discounttype) from discounts

3、子句
(1)基础查询
select * from authors
星号*告诉数据库返回由 from 指定的表的所有列，返回顺序由数据库决定。大小写不影响查询结果。
如果你只对某些列感兴趣，比如只想检索 au_lname, au_fname, phone, address
select au_lname,au_fname, phone,address from authors

SELECT select_list
[INTO new_table]
FROM table_source
[WHERE search_condition]
[GROUP BY group_by_expression]
[HAVING search_condition]
[ORDER BY order_expression [ASC | DESC]]

(2)where 子句
where <搜索条件>
例：
select * from authors

结果返回了数据库 pubs 中表 authors 的所有内容，如果你想查某一作者的情况，可以键入 ：
select * from authors where au_lname ='Smith'
(3)order by 子句
把查询结果按顺序显示，使用 order by 子句：
SELECT * from titles
记录是按录入顺序显示的。
按价格从小到大排序：
SELECT * from titles order by price
按价格从大到小排序：
SELECT * from titles order by price desc
从小到大是升序排列，关键字是 asc，系统默认排序为升序，所以 asc 可以省略。
从大到小是降序排列，关键字是 desc。

order by 子句也可以用来对字符型数据排序。
SELECT * from titles order by title asc
order by 子句也可以按多列进行排序：
SELECT * from titles order by title, type, price
注意优先级：
SELECT * from titles order by price, type, title

(4)group by 子句
指定用来放置输出行的组，并且如果 SELECT 子句 <select list> 中包含聚合函数，则计算每组的汇总值。
指定 group by 时，选择列表中任一非聚合表达式内的所有列都应包含在 group by 列表中，或者 group by 表达式必须与选择列表表达式完全匹配。

说明 如果未指定 ORDER BY 子句，则使用 GROUP BY 子句返回的组没有任何特定的顺序。
建议始终使用 ORDER BY 子句指定特定的数据顺序。

SELECT * from titleview
注意：titleview 是视图，视图的概念我们将在后面讲解
计算总价格：
SELECT sum(price) from titleview
我们使用多个聚集函数，按每个人分组计算价格：
SELECT au_lname, sum(price) 分组和, count(au_lname) 计数 from titleview group by au_lname

(5)子句组合
SELECT au_lname, sum(price) 分组和, count(au_lname) 计数 from titleview group by au_lname order by au_lname desc

(6)having 子句
having 子句用在给 group by 子句中的数据加限制条件。HAVING 通常与 GROUP BY 子句一起使用。
SELECT au_lname, sum(price) 分组和, count(au_lname) 计数 from titleview group by

au_lname having sum(price)>20 order by sum(price)

HAVING 可以让你在比较表达式中使用聚集函数，而 WHERE 中不能有聚合函数！！！

WHERE 和 ORDER BY 子句经常用于单行的查询，就像这样：

SELECT * from titleview where price>20 order by au_lname desc

GROUP BY 和 HAVING 一般用于合计，像上页的例子。

四、创建和操作表

1、创建数据库/删除数据库

2、创建表

```
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[学生档案]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[学生档案]
GO

CREATE TABLE [dbo].[学生档案] (
            [s_id] [int] IDENTITY (1, 1) NOT NULL ,
            [学号] [varchar] (10) COLLATE Chinese_PRC_CI_AS NULL ,
            [姓名] [varchar] (10) COLLATE Chinese_PRC_CI_AS NULL ,
            [性别] [char] (2) COLLATE Chinese_PRC_CI_AS NULL ,
            [班级] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[学生档案] WITH NOCHECK ADD CONSTRAINT [PK_学生档案] PRIMARY KEY
CLUSTERED
            (
                [s_id]
            )  ON [PRIMARY]
GO

ALTER TABLE [dbo].[学生档案] ADD
            CONSTRAINT [DF_学生档案_学号] DEFAULT ('') FOR [学号],
            CONSTRAINT [DF_学生档案_姓名] DEFAULT ('') FOR [姓名],
            CONSTRAINT [DF_学生档案_性别] DEFAULT ('') FOR [性别],
            CONSTRAINT [DF_学生档案_班级] DEFAULT ('') FOR [班级]
GO
```

“学生档案”的样本数据

学号	姓名	性别	班级
001	王小童	男	初二一班
002	张柳风	女	初二一班
003	紫云飞	女	初二三班
004	黄天龙	男	初二二班

五、连接(JOIN)

1、键、键列

键：唯一标识行 (主键 PRIMARY KEY)、定义两表之间的关系 (外键 FOREIGN KEY) 或用于生成索引的一个列或一组列。

键列：由主键、外键或索引键引用的列。

主键(PK)

唯一标识表中的所有行的一个列或一组列。主键不允许空值。

不能存在具有相同的主键值的两个行，因此主键值总是唯一标识单个行。

表中可以有不止一个键唯一标识行，每个键都称作候选键。

只有一个候选键可以选作表的主键，所有其它候选键称作备用键。

外键(FK)

列或列的组合，其值与同一个表或另一个表中的主键 (PK) 或唯一键相匹配。也称作参照键。

2、表关系类型

关系是通过匹配键列中的数据而工作的，而键列通常是两个表中具有相同名称的列。

在大多数情况下，关系**将一个表中为每个行提供唯一标识符的主键与另一个表中外键内的项**相匹配。

例如，通过在 titles 表的 title_id 列（主键）和 sales 表的 title_id 列（外键）之间创建一个关系，可以使销售额与特定的销售书名相关联。

表与表之间存在三种类型的关系。所创建的关系类型取决于相关联的列是如何定义的。

(1)一对一关系

在一对一关系中，表 A 中的一行最多只能与表 B 中的一行相匹配，反之亦然。如果两个相关列都是主键或具有唯一约束，则创建的是一对一关系。

这种关系不常见，因为这种方式的大部分相关信息都在一个表中。

(2)一对多关系

一对多关系是最常见的关系类型。

该关系中第一个表中的单个行可以与第二个表中的一个或多个行相关（匹配），但第二个表中的一个行只可以与第一个表中的一个行相关。

例如，publishers 表和 titles 表是一对多的关系：每一个出版商可出版许多书，但每一本书只能有一个出版商。

如果在相关列中只有一列是主键或具有唯一约束，则创建的是一对多关系。

(3)多对多关系

在多对多关系中，表 A 中的一行可与表 B 中的多行相匹配，反之亦然。

通过定义称为连接表的第三方表创建这样的关系，该连接表的主键包括表 A 和表 B 中的外键。

例如，authors 表和 titles 表是多对多关系，该关系通过从这些表中的每个表与 titleauthors 表的一对多关系定义。titleauthors 表的主键由 au_id 列（authors 表的主键）和 title_id 列（titles 表的主键）组成。

连接表：建立其它表之间的关系的表。连接表包含引用构成该关系的表的外键。

例如，OrderParts 连接表可以通过具有 Orders 表和 Parts 表的外键来展示每个订单运送的部件。

3、连接两个表

(1)交叉(CROSS)连接

如果我们需要的信息来自两个表，可以这样：

SELECT dbo.学生档案.学号, dbo.学生成绩.学习科目 FROM dbo.学生成绩 CROSS JOIN dbo.学生档案

(2)内联接(...INNER JOIN ...ON...)

通过比较源表间共享的列的值从多个源表检索行的操作。

例：

SELECT dbo.学生档案.学号, dbo.学生档案.姓名, dbo.学生档案.性别, dbo.学生档案.班级, dbo.学生成绩.学习科目, dbo.学生成绩.学习成绩 FROM **dbo.学生成绩 INNER JOIN dbo.学生档案 ON dbo.学生成绩.s_id = dbo.学生档案.s_id**

PPT 写法不推荐

SELECT 学生档案.学号，学生成绩.科目代码，学生成绩.成绩 FROM 学生档案，学生成绩 where 学生档案.学号=学生成绩.学号

如果想加入课程表：（n 各表相连，where 中有 n-1 个等号）

SELECT 学生档案.学号，学生成绩.科目代码，学生成绩.成绩 课程表.科目名称 FROM 学生档案，学生成绩， 课程表 where 学生档案.学号=学生成绩.学号 and 学生成绩.科目代码=课程表.科目代码

将一个表连接到它本身

将一个表连接到它自身是个经常使用的技术，**用来找出有重复字段的记录**。

例：(等值联接与非等值联接)

SELECT dbo.学生档案.学号, dbo.学生档案.姓名 FROM dbo.学生档案 INNER JOIN dbo.学生档案 学生档案_1 ON dbo.学生档案.姓名 = 学生档案_1.姓名 AND dbo.学生档案.学号 <> 学生档案_1.学号

(3)外联接

在 s_id 列上联接 学生档案 表和 学生成绩 表，结果只显示相匹配的数据。

左连接。

不管表二中是否有匹配的数据，结果将包含表一中的所有行。

SELECT <字段名> FROM <表 1> LEFT OUTER JOIN <表 2> <ON 子句>

例：

若要在结果中包括所有的学生信息，而不管 学生成绩 表中是否有关联的记录，可以使用左向外联接 LEFT OUTER JOIN 。

SELECT dbo.学生档案.学号, dbo.学生档案.姓名, dbo.学生档案.性别, dbo.学生档案.班级,dbo.学生成绩.学习科目, dbo.学生成绩.学习成绩, dbo.学生成绩.s_id FROM **dbo.学生档案 LEFT OUTER JOIN dbo.学生成绩** ON dbo.学生档案.s_id = dbo.学生成绩.s_id

右连接

不管表一中是否有匹配的数据，结果将包含表二中的所有行。

SELECT <字段名> FROM <表 1> RIGHT OUTER JOIN <表 2> <ON 子句>
例：
若要在结果中包括所有的学生成绩，而不管 学生档案 表中是否有关联的记录，可以使用右向外联接运算符 RIGHT OUTER JOIN。
SELECT dbo.学生档案.学号, dbo.学生档案.姓名, dbo.学生档案.性别, dbo.学生档案.班级,dbo.学生成绩.学习科目, dbo.学生成绩.学习成绩, dbo.学生成绩.s_id FROM dbo.学生档案 RIGHT OUTER JOIN dbo.学生成绩 ON dbo.学生档案.s_id = dbo.学生成绩.s_id

全连接
不管表中是否有匹配的数据，结果将包括两个表中的所有行。
SELECT <字段名> FROM <表 1> FULL OUTER JOIN <表 2> <ON 子句>
例：
SELECT dbo.学生档案.学号, dbo.学生档案.姓名, dbo.学生档案.性别, dbo.学生档案.班级, dbo.学生成绩.学习科目, dbo.学生成绩.学习成绩, dbo.学生成绩.s_id FROM dbo.学生档案 **FULL OUTER JOIN** dbo.学生成绩 ON dbo.学生档案.s_id = dbo.学生成绩.s_id

六、数据操纵语句
(1)INSERT 语句
INSERT INTO 表名(列 1，列 2…) VALUES (值 1，值 2…)
INSERT INTO 学生档案 (学号,姓名,性别) VALUES ('009','李刚','男')
一次插入一条记录

INSERT…SELECT
通过一条 sql 语句实现从多个表中组合字段然后插入到另外的一个新表中。
两表之间：
INSERT INTO tab1(field1,field2) SELECT field1,field2 FROM tab2;
多表之间：

```
INSERT INTO tab1 (field1, field2, field3, ... )
  SELECT * FROM
  (SELECT tab2.field1, tab3.field2, tab4.field3,  ...
  FROM tab2
  JOIN tab3 ON tab2.field = tab3.filed
  JOIN tab4 ON tab2.field = tab4.filed
    ...
    ) AS tab ;
```

(2)UPDATE 语句
UPDATE 语句用于改变现有记录中字段的值。
UPDATE 表名 SET 列 1=值 1,列 2=值 2 WHERE 搜索条件
例：
UPDATE 学生档案 SET 班级='初二一班'
注意：因为省略了 WHERE 子句，表中的每条记录的相应字段都被更新。
所以，一般情况下 UPDATE 语句应带 WHERE 子句。
例：

UPDATE 学生档案 SET 班级='初二三班' WHERE s_id=2

(3)DELETE 语句
DELETE 语句从表中删除记录。
DELETE FROM 表名 WHERE 条件
例：
DELETE FROM 学生档案
注意：和 UPDATE 一样，如果省略了 WHERE 子句，表中的所有记录将被删除。
所以，一般情况下 DELETE 语句应带 WHERE 子句。
例：
DELETE FROM 学生档案 WHERE s_id=2

上课时记的笔记

表的内联

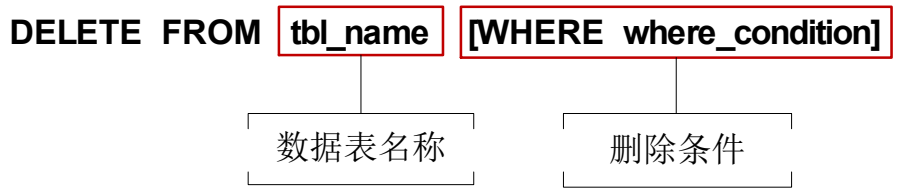
SELECT 学生档案.学号, 学生成绩.科目代码，学生成绩.成绩 FROM 学生档案，学生成绩 where 学生档案.学号=学生成绩.学号

如果想加入课程表：（n 各表相连，where 中有 n-1 个等号）

SELECT 学生档案.学号, 学生成绩.科目代码，学生成绩.成绩 课程表.科目名称 FROM 学生档案，学生成绩， 课程表 where 学生档案.学号=学生成绩.学号 and 学生成绩.科目代码=课程表.科目代码

Update、删除信息来源于多个表

Update php_admin set userage=41 (FROM 多个表）where （多个表时要进行表的内联）username = '吕布'
具体例子见这一部分例题



信息来源于多个表时，不能用 from 表 1，表 2
Delete 目标表 from 表 1，表 2… where 内联加索引，跟 update 很像