MySQL必知必会

选择数据库：USE sure;

检索

检索单列：`SELECT prod_name from products;`
检索多列：`SELECT prod_name, prod_id, prod_price FROM products;`
检索全部列：`SELECT * FROM products;`
检索不同列：`SELECT DISTINCT vend_id FROM products;`
限制结果：`SELECT prod_name FROM products LIMIT 5;`
`SELECT prod_name FROM products LIMIT 5,5;`
使用完全限定表名：`SELECT products.prod_name FROM products;`
`SELECT products.prod_name FROM sure.products;`

排序

排序数据：`SELECT prod_name FROM products ORDER BY prod_name;`
按多个列排序：`SELECT prod_id, prod_price, prod_name FROM products ORDER BY prod_price, prod_name;`
指定排序方向：`SELECT prod_id, prod_price, prod_name FROM products ORDER BY prod_price DESC;`
多个列排序：`SELECT prod_id, prod_price, prod_name FROM products ORDER BY prod_price DESC, prod_name;`
`prod_price`是按降序，但`prod_name`还是按默认升序排的。
如果想在多个列上进行降序排序，必须对每个列指定`DESC`关键字。

使用ORDER BY和LIMIT的组合，能够找出一个列中最高或最低的值。
`SELECT prod_price FROM products ORDER BY prod_price DESC LIMIT 1;`

在给出ORDER BY子句时，应该保证它位于FROM子句之后。如果使用LIMIT，它必须位于ORDER BY之后。使用子句的次序不对将产生错误消息。

过滤数据

使用WHERE子句： SELECT prod_price, prod_name FROM products WHERE prod_price = 2.5;
WHERE子句的位置 在同时使用ORDER BY和WHERE子句时，应该让ORDER BY位于WHERE之后，否则将会产生错误。
WHERE语句操作符：=, !=, <>, >, >=, <, <=, BETWEEN
范围值检查：SELECT prod_name, prod_price FROM products WHERE prod_price BETWEEN 5 AND 10;
空值检查：SELECT prod_name FROM products WHERE prod_price IS NULL;
NULL与不匹配 在通过过滤选择出不具有特定值的行时，你可能希望返回具有NULL值的行。但是，不行。因为未知具有特殊的含义，数据库不知道它们是否匹配，所以在匹配过滤或不匹配过滤时不返回它们。因此，在过滤数据时，一定要验证返回数据中确实给出了被过滤列具有NULL的行。

数据过滤 - 组合WHERE子句

组合WHERE子句：
AND操作符： SELECT prod_id, prod_price, prod_name FROM products WHERE vend_id = 1003 AND prod_price <= 10;
OR操作符：SELECT prod_name, prod_price FROM products WHERE vend_id = 1002 OR vend_id = 1003;
IN操作符：IN操作符用来指定条件范围，范围中的每个条件都可以进行匹配。IN取合法值的由逗号分隔的清单，全都括在圆括号中。
SELECT prod_price, prod_name FROM products WHERE vend_id IN (1002, 1003) ORDER BY prod_name;
为什么要使用IN操作符？其优点具体如下：
❑ 在使用长的合法选项清单时，IN操作符的语法更清楚且更直观。
❑ 在使用IN时，计算的次序更容易管理（因为使用的操作符更少）。
❑ IN操作符一般比OR操作符清单执行更快。
❑ IN的最大优点是可以包含其他SELECT语句，使得能够更动态地建立WHERE子句。第14章将对此进行详细介绍。
IN WHERE子句中用来指定要匹配值的清单的关键字，功能与OR相当。
NOT操作符：WHERE子句中的NOT操作符有且只有一个功能，那就是否定它之后所跟的任何条件。
SELECT prod_price, prod_name FROM products WHERE vend_id NOT IN (1002, 1003) ORDER BY prod_name;

用通配符进行过滤

LIKE指示MySQL，后跟的搜索模式利用通配符匹配而不是直接相等匹配进行比较。
%操作符：%表示任何字符出现任意次数。为了找出所有以词jet起头的产品，可使用以下SELECT语句：
SELECT prod_id, prod_name FROM products WHERE prod_name LIKE 'jet%';
SELECT prod_id, prod_name FROM products WHERE prod_name LIKE '%anvil%';
SELECT prod_name FROM products WHERE prod_name LIKE 's%e';
重要的是要注意到，除了一个或多个字符外，%还能匹配0个字符。%代表搜索模式中给定位置的0个、1个或多个字符。
注意NULL 虽然似乎%通配符可以匹配任何东西，但有一个例外，即**NULL**。即使是WHERE prod_name LIKE '%'也**不能匹配**用值NULL作为产品名的行。

_通配符：下划线只匹配单个字符而不是多个字符。

正则表达式

LIKE匹配整个列。如果被匹配的文本在列值中出现，LIKE将不会找到它，相应的行也不被返回（除非使用通配符）。而REGEXP在列值内进行匹配，如果被匹配的文本在列值中出现，REGEXP将会找到它，相应的行将被返回。这是一个非常重要的差别。

计算字段

在上述每个例子中，存储在表中的数据都不是应用程序所需要的。我们需要直接从数据库中检索出转换、计算或格式化过的数据；而不是检索出数据，然后再在客户机应用程序或报告程序中重新格式化。这就是计算字段发挥作用的所在了。与前面各章介绍过的列不同，计算字段并不实际存在于数据库表中。计算字段是运行时在SELECT语句内创建的。
拼接字段： 使用AS当作别名，SELECT Concat(vend_name, '(', vend_country, ')') AS vend_title FROM vendors ORDER BY vend_name;
执行算术计算。

汇总数据

聚集函数：AVG(), COUNT(), MAX(), MIN(), SUM()
(1) AVG: SELECT AVG(prod_price) AS avg_price FROM products;
只用于单个列 AVG()只能用来确定特定数值列的平均值，而且列名必须作为函数参数给出。为了获得多个列的平均值，必须使用多个AVG()函数。
(2) COUNT:
COUNT()函数有两种使用方式。
❑ 使用COUNT(*)对表中行的数目进行计数，不管表列中包含的是空值（NULL）还是非空值。
❑ 使用COUNT(column)对特定列中具有值的行进行计数，忽略NULL值。
SELECT COUNT(*) AS num_cust FROM customers;
SELECT COUNT(cust_email) AS num_cust FROM customers;
(3) MAX: SELECT MAX(prod_price) AS max_price FROM products;
(4) MIN: SELECT MIN(prod_price) AS min_price FROM products;
(5) SUM: SELECT SUM(quantity) AS items_ordered FROM orderitems WHERE order_num = 20005;

以上5个聚集函数都可以如下使用：
❑ 对所有的行执行计算，指定ALL参数或不给参数（因为ALL是默认行为）；
❑ 只包含不同的值，指定DISTINCT参数。
SELECT AVG(DISTINCT prod_price) AS avg_price FROM products WHERE vend_id = 1003;
如果指定列名，则DISTINCT只能用于COUNT()。DISTINCT不能用于COUNT(*)，因此不允许使用COUNT（DISTINCT），否则会产生错误。类似地，DISTINCT必须使用列名，不能用于计算或表达式。

组合聚集函数：

```
SELECT COUNT(*) AS num_items,
       MIN(prod_price) AS price_min,
       MAX(prod_price) AS price_max,
       AVG(prod_price) AS price_avg
FROM products;
```

分组数据

分组允许把数据分为多个逻辑组，以便能对每个组进行聚集计算。
创建分组： SELECT vend_id, COUNT(*) AS num_prods FROM products GROUP BY vend_id;
在具体使用GROUP BY子句前，需要知道一些重要的规定。
❑ GROUP BY子句可以包含任意数目的列。这使得能对分组进行嵌套，为数据分组提供更细致的控制。
❑ 如果在GROUP BY子句中嵌套了分组，数据将在最后规定的分组上进行汇总。换句话说，在建立分组时，指定的所有列都一起计算（所以不能从个别的列取回数据）。
❑ GROUP BY子句中列出的每个列都必须是检索列或有效的表达式（但不能是聚集函数）。如果在SELECT中使用表达式，则必须在GROUP BY子句中指定相同的表达式。不能使用别名。
❑ 除聚集计算语句外，SELECT语句中的每个列都必须在GROUP BY子句中给出。
❑ 如果分组列中具有NULL值，则NULL将作为一个分组返回。如果列中有多行NULL值，它们将分为一组。
❑ GROUP BY子句必须出现在WHERE子句之后，ORDER BY子句之前
过滤分组

唯一的差别是WHERE过滤行，而HAVING过滤分组。
SELECT cust_id, COUNT(*) AS orders FROM orders GROUP BY cust_id HAVING COUNT(*) >= 2;
HAVING和WHERE的差别 这里有另一种理解方法，WHERE在数据分组前进行过滤，HAVING在数据分组后进行过滤。这是一个重要的区别，WHERE排除的行不包括在分组中。这可能会改变计算值，从而影响

HAVING子句中基于这些值过滤掉的分组。
在一条语句中同时使用WHERE和HAVING子句, 列出具有2个（含）以上、价格为10（含）以上的产品的供应商：

SELECT vend_id, COUNT(*) AS num_prods FROM products WHERE prod_price >= 10 GROUP BY vend_id HAVING COUNT(*) >= 2;

SELECT子句顺序：

子　句	说　明	是否必须使用
SELECT	要返回的列或表达式	是
FROM	从中检索数据的表	仅在从表选择数据时使用
WHERE	行级过滤	否
GROUP BY	分组说明	仅在按组计算聚集时使用
HAVING	组级过滤	否
ORDER BY	输出排序顺序	否
LIMIT	要检索的行数	否

子查询

```
SELECT cust_name, cust_contact
FROM customers
WHERE cust_id IN (SELECT cust_id
                  FROM orders
                  WHERE order_num IN (SELECT order_num
                                      FROM orderitems
                                      WHERE prod_id = 'TNT2'));
```

```
SELECT cust_name,
       cust_state,
       (SELECT COUNT(*)
        FROM orders
        WHERE orders.cust_id = customers.cust_id) AS orders
FROM customers
ORDER BY cust_name;
```

子查询最常见的使用是在WHERE子句的IN操作符中，以及用来填充计算列。
“今天本来有点疑惑，方法一用了临时表为什么就能当有一条记录时提出NULL，后来看到评论里有提到这是子查询的特质 子查询数据出虚表嵌套查询虚表,如果查询不到会返回null,棒”

联结表

外键为某个表中的一列，它包含另一个表的主键值，定义了两个表之间的关系。

```
SELECT vend_name, prod_name, prod_price
FROM vendors, products
WHERE vendors.vend_id = products.vend_id
ORDER BY vend_name, prod_name;
```

内部联结

```
SELECT vend_name, prod_name, prod_price
FROM vendors INNER JOIN products
 ON vendors.vend_id = products.vend_id;
```

联结多个表：

```
SELECT prod_name, vend_name, prod_price, quantity
FROM orderitems, products, vendors
WHERE products.vend_id = vendors.vend_id
  AND orderitems.prod_id = products.prod_id
  AND order num = 20005;
```

高级联结

表别名，可以缩短SQL语句 AND 允许在单条SELECT语句中多次使用相同的表
使用不同类型的联结：自联结、自然联结和外部联结
自联结：SELECT p1.prod_id, p1.prod_name FROM products AS p1, products AS p2 WHERE p1.vend_id = p2.vend_id AND p2.prod_id = 'DTNTR';
用自联结而不用子查询 自联结通常作为外部语句用来替代从相同表中检索数据时使用的子查询语句。虽然最终的结果是相同的，但有时
候处理联结远比处理子查询快得多。

无论何时对表进行联结，应该至少有一个列出现在不止一个表中（被联结的列）。标准的联结（前一章中介绍的内部联结）返回所有数据，甚至相同的列多次出现。自然联结排除多次出现，使每个列只返回一次。

外部联结：在上述例子中，联结包含了那些在相关表中没有关联行的行。这种类型的联结称为外部联结。
SELECT customers.cust_id, orders.order_num FROM customers LEFT OUTER JOIN orders ON customers.cust_id = orders.cust_id;
但是，与内部联结关联两个表中的行不同的是，外部联结还包括没有关联行的行。在使用OUTER JOIN语法时，必须使用RIGHT或LEFT关键字指定包括其所有行的表（RIGHT指出的是OUTER JOIN右边的表，而LEFT指出的是OUTER JOIN左边的表）

在总结关于联结的这两章前，有必要汇总一下关于联结及其使用的某些要点。
- ❑ 注意所使用的联结类型。一般我们使用内部联结，但使用外部联结也是有效的。
- ❑ 保证使用正确的联结条件，否则将返回不正确的数据。
- ❑ 应该总是提供联结条件，否则会得出笛卡儿积。
- ❑ 在一个联结中可以包含多个表，甚至对于每个联结可以采用不同的联结类型。虽然这样做是合法的，一般也很有用，但应该在一起测试它们前，分别测试每个联结。这将使故障排除更为简单。

组合查询

UNION与WHERE 本章开始时说过，UNION几乎总是完成与多个WHERE条件相同的工作。UNION ALL为UNION的一种形式，它完成WHERE子句完成不了的工作。如果确实需要每个条件的匹配行全部出现（包括重复行），则必须使用UNION ALL而不是WHERE。

更新和删除

客户10005现在有了电子邮件地址，因此他的记录需要更新，语句如下：
`UPDATE customers SET cust_name = 'The Fudds', cust_email = 'elmer@fudd.com' WHERE cust_id = 10005;`
删除一行：`DELETE FROM customers WHERE cust_id = 10006;`

DELETE不需要列名或通配符。DELETE删除整行而不是删除列。为了删除指定的列，请使用UPDATE语句。

创建和操纵表

如果主键使用单个列，则它的值必须唯一；如果主键包含多个列，
每个表只允许一个AUTO_INCREMENT列，而且它必须被索引（如，通过使它成为主键）。

更新表：`ALTER TABLE vendors ADD vend_phone CHAR(20);`
`ALTER TABLE vendors DROP COLUMN vend_phone;`
ALTER TABLE的一种常见用途是定义外键。

```
ALTER TABLE orderitems
ADD CONSTRAINT fk_orderitems_orders
FOREIGN KEY (order_num) REFERENCES orders (order_num);

ALTER TABLE orderitems
ADD CONSTRAINT fk_orderitems_products FOREIGN KEY (prod_id)
REFERENCES products (prod_id);

ALTER TABLE orders
ADD CONSTRAINT fk_orders_customers FOREIGN KEY (cust_id)
REFERENCES customers (cust_id);

ALTER TABLE products
ADD CONSTRAINT fk_products_vendors
FOREIGN KEY (vend_id) REFERENCES vendors (vend_id);
```

视图

这就是视图的作用。productcustomers是一个视图，作为视图，它不包含表中应该有的任何列或数据，它包含的是一个SQL查询（与上面用以正确联结表的相同的查询）。
下面是视图的一些常见应用。

- ❑ 重用SQL语句。
- ❑ 简化复杂的SQL操作。在编写查询后，可以方便地重用它而不必知道它的基本查询细节。
- ❑ 使用表的组成部分而不是整个表。
- ❑ 保护数据。可以给用户授予表的特定部分的访问权限而不是整个表的访问权限。
- ❑ 更改数据格式和表示。视图可返回与底层表的表示和格式不同的数据。

重要的是知道视图仅仅是用来查看存储在别处的数据的一种设施。视图本身不包含数据，因此它们返回的数据是从其他表中检索出来的。在添加或更改这些表中的数据时，视图将返回改变过的数据。
在理解什么是视图（以及管理它们的规则及约束）后，我们来看一下视图的创建。

- ❑ 视图用CREATE VIEW语句来创建。
- ❑ 使用SHOW CREATE VIEW viewname；来查看创建视图的语句。
- ❑ 用DROP删除视图，其语法为DROP VIEW viewname;。
- ❑ 更新视图时，可以先用DROP再用CREATE，也可以直接用CREATE ORREPLACE VIEW。如果要更新

的视图不存在，则第2条更新语句会创建一个视图；如果要更新的视图存在，则第2条更新语句会替换原有视图。

```
CREATE VIEW productcustomers AS
SELECT cust_name, cust_contact, prod_id
FROM customers, orders, orderitems
WHERE customers.cust_id = orders.cust_id
  AND orderitems.order_num = orders.order_num;
```

一些技巧

使用 OR 会使索引会失效，在数据量较大的时候查找效率较低，通常建议使用 union 代替 or。

知道使用group by和having。还需要记得优先顺序。where > group by > having > order by