

1 2检索数据

1.1 检索不同的值

结果去重，查询列的所有取值

```
1 | SELECT DISTINCT xxx from xxx;
```

1.2 限制结果

取前五名

SQL Server:

```
1 | SELECT TOP 5 xxx from xxx;
```

MySQL:

```
1 | SELECT xxx from xxx LIMIT 5;
```

切片

```
1 | SELECT xxx from xxx LIMIT 5 OFFSET 5;
```

或者再MySQL中使用快捷短语

```
1 | SELECT xxx from xxx LIMIT 5,5;
```

返回从第5行起的5行数据。第一个数字是开始位置，第二个数字是检索行数

[注]: 角标从第0行开始

1.3 使用注释

1.3.1 行内注释

```
1 SELECT xxx -- 这是一行注释
2 from xxx;
```

1.3.2 整行注释

```
1 # 这是一行注释
2 SELECT xxx from xxx;
```

1.3.3 多行注释

```
1 /* SELECT xxx
2 FROM xxx; */
3 SELECT xxx from xxx;
```

2 排序

2.1 单列排序

按照 prod_name 的字母顺序排序

```
1 SELECT prod_name
2 FROM Product
3 ORDER BY prod_name;
```

[注]: 确保 ORDER BY 子句是最后一行, 否则会报错

2.2 多行排序

比如员工表，需要按照姓和名排序，如果同姓，则在同姓之间再按照名排序。

```
1 SELECT prod_id, prod_price, prod_name
2 FROM Products
3 ORDER BY prod_price, prod_name;
```

[注]: 仅在多行 `prod_price` 相同的时候再按照 `prod_name` 排序。

2.3 按列位置排序

```
1 SELECT prod_id, prod_prices, prod_name
2 FROM Products
3 ORDER BY 2, 3;
```

`ORDER BY 2, 3` 表示先按 `prod_price`，再按 `prod_name` 进行排序。

2.4 指定方向排序

默认排序方式是从A-Z，使用 `DESC` 关键字可以降序排序。

```
1 SELECT prod_id, prod_prices, prod_name
2 FROM Products
3 ORDER BY prod_price DESC;
```

如果打算选出最贵的，再加上产品名

```
1 SELECT prod_id, prod_prices, prod_name
2 FROM Products
3 ORDER BY prod_prices DESC, prod_names;
```

3 过滤数据

3.1 使用where语句

```
1 SELECT prod_name, prod_price
2 FROM Products
3 WHERE prod_price = 3.49;
```

操作符	说明
!=	不等于
BETWEEN x AND y	在指定2个值之间
IS NULL	为NULL

4 高级数据过滤

4.1 组合where子句

4.1.1 AND操作符

```
1 SELECT prod_id, prod_price, prod_name
2 FROM Products
3 WHERE vend_id = 'DLL01' AND prod_price <=4 ;
```

4.1.2 OR操作符

```
1 SELECT prod_name, prod_price
2 FROM Products
3 WHERE vend_id = 'DLL01' OR vend_id = 'BRS01';
```

4.1.3 求值顺序

使用 () 确定 OR AND 操作顺序

```
1 SELECT prod_name, prod_price
2 FROM Products
3 WHERE (vend_id = 'DLL01' OR vend_id = 'BRS01')
4 AND prod_price >= 10;
```

4.2 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。

```
1 SELECT prod_name, prod_price
2 FROM Products
3 WHERE vend_id IN ( 'DLL01', 'BRS01')
4 ORDER BY prod_names;
```

4.3 NOT操作符

`WHERE` 子句中的 `NOT` 操作符有且只有一个功能，那就是否定其后所跟的任何条件。

```
1 SELECT prod_name
2 FROM Products
3 WHERE NOT vend_id = 'DLL01'
4 ORDER BY prod_name;
```

5 使用通配符进行过滤

5.1 LIKE操作符

5.1.1 %通配符

`%`表示任何字符出现任意次数。

例如，为了找出所有以词 `Fish` 起头的产品，可发布以下 `SELECT` 语句：

```
1 SELECT prod_id, prod_name
2 FROM Products
3 WHERE prod_name LIKE 'Fish%';
```

5.1.2 _通配符

和%不同之处在于，_只匹配一次。

```
1 SELECT prod_id, prod_name
2 FROM Products
3 WHERE prod_name LIKE '__ inch teddy bear';
```

输出▼

1	prod_id	prod_name
2	-----	-----
3	BR02	12 inch teddy bear
4	BR03	18 inch teddy bear

分析▼

这个WHERE子句中的搜索模式给出了后面跟有文本的两个通配符。结果只显示匹配搜索模式的行：第一行中下划线匹配12，第二行中匹配18。8 inch teddy bear 产品没有匹配，因为搜索模式要求匹配两个通配符而不是一个。

5.1.3 []通配符

方括号（[]）通配符用来指定一个字符集，它必须匹配指定位置（通配符的位置）的一个字符。

例如，找出所有名字以J或M起头的联系人，可进行如下查询：

```
1 SELECT cust_contact
2 FROM Customers
3 WHERE cust_contact LIKE '[JM]%'
4 ORDER BY cust_contact;
```

此通配符可以用前缀字符^（脱字号）来否定。例如，下面的查询匹配以J和M之外的任意字符起头的任意联系人名（与前一个例子相反）：

```
1 SELECT cust_contact
2 FROM Customers
3 WHERE cust_contact LIKE '[^JM]%'
4 ORDER BY cust_contact;
```

6 创建计算字段

6.1 拼接字段

```
1 SELECT vend_name || '(' || vend_country || ')'
2 FROM Venders
3 ORDER BY vend_name
```

输出▼

```
1 Bear Emporium          (USA )
2 Bears R Us             (USA )
3 Doll House Inc.        (USA )
4 Fun and Games          (England )
5 Furball Inc.           (USA )
6 Jouets et ours         (France )
```

如果使用MySQL，则使用以下语法

```
1 SELECT Concat(vend_name, '(', vend_country, ')')
2 FROM Vendors
3 ORDER BY vend_name;
```

以上输出包含空格，如果不需要这些空格，可以使用 `RTRIM()` 函数。

```
1 SELECT RTRIM(vend_name) + '(' + RTRIM(vend_country) +
   ')'
2 FROM Vender
3 ORDER BY vend_name;
```

输出▼

```
1 Bear Emporium (USA)
2 Bears R Us (USA)
3 Doll House Inc. (USA)
4 Fun and Games (England)
5 Furball Inc. (USA)
6 Jouets et ours (France)
```

[注]: `RTRIM()` 删除字符串右边的所有空格。 `LTRIM()` 删除左边空格，`TRIM()` 删除两侧空格

6.1.1 使用别名

前面的SELECT语句很好地拼接了字段，但是仅仅是输出，不能通过 `SELECT` 查询。

```
1 SELECT RTRIM(vend_name) + '(' + RTRIM(vend_country) +
   ')' AS vend_title
2 FROM Vendors
3 ORDER BY vend_name;
```


如果使用MySQL，则使用：

```
1 SELECT Concat(RTRIM(vend_name), '(',  
2 RTRIM(vend_country), ')') AS vend_title  
3 ORDER BY vend_name;
```

6.2 执行算数计算

```
1 SELECT prod_id, quantity, item_price,  
2 quantity*item_price AS expanded_price  
3 FROM OrderItems  
4 WHERE order_num = 20008;
```

7 使用函数

7.1 文本处理函数

函数	说明
LENGTH()	返回字符串长度
LOWER()/UPPER()	大小写转换
SUBSTR()	字符串切片

7.1.1 SOUNDEX()

SOUNDEX() 是将任何文本串转换为描述其语音的算法。SOUNDEX() 考虑了类似的发音。

```
1 SELECT cust_name, cust_contact  
2 FROM Customers  
3 WHERE SOUNDEX(cust_contact) = SOUNDEX('Michael Green')
```

7.2 日期和时间处理函数

```
1 SELECT order_num
2 FROM orders
3 WHERE YEAR(order_date) = 2020
```

8 汇总数据

8.1 汇聚函数

8.1.1 AVG()

[注]: `AVG()` 只作用单行，且忽略值为 `NULL` 的行。

```
1 SELECT AVG(prod_price) AS avg_price
2 FROM Products
3 WHERE vend_id = 'DLL01';
```

输出▼

```
1 avg_price
2 -----
3 3.8650
```

8.1.2 COUNT()

1. `COUNT(*)` 不会忽略 `NULL`，默认不统计 `NULL`
2. 使用 `COUNT(Column)` 对特定列有取值的行数计数，忽略 `NULL`

```
1 SELECT COUNT(*) AS num_cust
2 FROM Customers;
```

如果只统计填写电子邮件的客户：

```
1 | SELECT COUNT(cust_email) AS num_cust
2 | FROM Customers;
```

8.1.3 MAX()

如果应用于字符串列，返回排序后的最后一行。

8.2 组合聚集函数

```
1 | SELECT COUNT(*) AS num_items,
2 |         MIN(prod_price) AS price_min,
3 |         MAX(prod_price) AS price_max,
4 |         AVG(prod_price) AS price_avg
5 | FROM Products;
```

9 分组数据

9.1 创建分组

```
1 | SELECT vend_id, COUNT(*) AS num_prods
2 | FROM Products
3 | GROUP BY vend_id;
```

输出▼

vend_id	num_prods
-----	-----
BRS01	3
DLL01	4
FNG01	2

9.2 过滤分组

[注]: 所有的 where 都可以用 having 替代

```
1 SELECT cust_id, Count(*) AS orders
2 FROM Orders
3 GROUP BY coust_id
4 HAVING COUNT(*) >= 2;
```

输出▼

cust_id	orders
-----	-----
10000000001	2

9.2.1 分析

1. 最后一行 HAVING 语句过滤了2个订单以上的分组。

2. 这种情况下 `where` 不起作用，因为 `where` 只对行生效，`having` 对分组进行过滤。
3. `where` 在分组前进行过滤，`having` 在分组后进行过滤。

10 子查询

```
1 SELECT cust_id
2 FROM Orders
3 WHERE order_num IN (SELECT order_num
4                     FROM OrderItems
5                     WHERE prod_id = 'RGAN01');
```

[注]: 作为子查询的 `SELECT` 语句只能查询单个列。

11 联结

```
1 SELECT vend_name, prod_name, prod_price
2 FROM Vendors, Products
3 WHERE Vendors.vend_id = Products.vend_id;
```

输出▼

vend_name	prod_name	prod_price
Doll House Inc.	Fish bean bag toy	3.4900
Doll House Inc.	Bird bean bag toy	3.4900
Doll House Inc.	Rabbit bean bag toy	3.4900
Bears R Us	8 inch teddy bear	5.9900
Bears R Us	12 inch teddy bear	8.9900
Bears R Us	18 inch teddy bear	11.9900
Doll House Inc.	Raggedy Ann	4.9900
Fun and Games	King doll	9.4900
Fun and Games	Queen doll	9.4900

11.1 联结多个表

```

1 SELECT prod_name, vend_name, prod_price, quantity
2 FROM OrderItems, Products, Vendors
3 WHERE Products.vend_id = Vendors.vend_id
4       AND OrderItems.prod_id = Products.prod_id
5       AND order_num = 20007;

```

输出▼

prod_name	vend_name	prod_price	quantity
-----	-----	-----	-----
18 inch teddy bear	Bears R Us	11.9900	50
Fish bean bag toy	Doll House Inc.	3.4900	100
Bird bean bag toy	Doll House Inc.	3.4900	100
Rabbit bean bag toy	Doll House Inc.	3.4900	100
Raggedy Ann	Doll House Inc.	4.9900	50

12 创建高级联结

12.1 外联结

左外联结：以左表为基础，显示所有左表的行，对右表的行只显示符合条件的行。

```

1 SELECT Customers.cust_id, Orders.order_num
2 FROM Customers RIGHT OUTER JOIN Orders On
   Customers.cust_id = Orders.cust_id

```

13 组合查询

13.1 UNION操作符

```
1 SELECT cust_name, cust_contact, cust_email
2 FROM Customers
3 WHERE cust_state IN ('IL','IN','MI')
4 UNION
5 SELECT cust_name, cust_contact, cust_email
6 FROM Customers
7 WHERE cust_name = 'Fun4All';
```

[注]: `UNION` 默认去重, 如果需要返回所有匹配行, 包含重复的, 需要使用 `UNION ALL`

14 插入数据

14.1 插入完整的行

```
1 INSERT INTO customers
2 VALUES(1,
3         'string',
4         'CA',
5         NULL);
```

这种方法简单但是不安全, 如果使用更安全的方式, 可以使用以下较为繁琐的方法

14.2 插入部分行

正如所述, 使用 `INSERT` 的推荐方法是明确给出表的列名。使用这种语法, 还可以省略列, 这表示可以只给某些列提供值, 给其他列不提供值。

```
1 INSERT INTO customers(cust_id,
2                        cust_name,
3                        cust_address,
4                        cust_city)
5 VALUES(1,
6         'string',
7         NULL,
8         'CA');
```

14.3 插入检索出的数据

```
1 INSERT INTO customers(cust_id,  
2                        cust_contact,  
3                        cust_email,  
4                        cust_name)  
5 SELECT cust_id, cust_contact, cust_email, cust_name  
6 FROM CustNew;
```

注: `INSERT` 只插入一行, `INSERT SELECT` 可以插入多行

14.4 从一个表复制到另一个表

```
1 CREATE TABLE CustCopy AS SELECT * FROM Customers;
```

15 更新和删除数据

15.1 UPDATE 操作符

```
1 UPDATE Customers  
2 SET cust_email = 'kim@thetoystore.com'  
3     cust_contact = 'Kim'  
4 WHERE cust_id = 10000000005;
```

15.2 DELETE 语句

```
1 DELETE FROM Customers  
2 WHERE cust_id = 10000000006;
```


15.3 注意事项

1. `DELETE` 删除整行，`FROM` 可以根据DBMS省略。
2. 除非确实打算更新和删除每一行，否则绝对不要使用不带 `WHERE` 子句的 `UPDATE` 或 `DELETE` 语句。
3. 在 `UPDATE` 和 `DELETE` 前使用 `SELECT` 进行测试，保证过滤的是正确的记录。
4. SQL没有undo按钮，应该非常小心的使用 `UPDATE` 和 `DELETE`。

16 创建和操纵表

16.1 表创建基础

```
1 CREATE TABLE Products
2 (
3     prod_id CHAR(10) NOT NULL,
4     vend_id CHAR(10) NOT NULL,
5     prod_name CHAR(254) NOT NULL,
6     prod_price DECIMAL(8,2) NOT NULL,
7     prod_desc VARCHAR(1000) NULL
8 );
```

16.2 使用默认值

如果插入行的时候不给定值，则使用默认值。

```
1 CREATE TABLE OrderItems
2 (
3     order_num INTEGER NOT NULL,
4     order_item INTEGER NOT NULL,
5     prod_id CHAR(10) NOT NULL PRIMARY KEY, -- 定义主键
6     quantity INTEGER NOT NULL DEFAULT 1,
7     item_price DECIMAL(8,2) NOT NULL
8 );
```

16.3 更新表

16.3.1 增加新列

```
1 ALTER TABLE vendors
2 ADD vend_phone CHAR(20);
```

16.3.2 删除列

```
1 ALTER TABLE vendors
2 DROP COLUMN vend_phone;
```

16.3.3 删除整个表

```
1 DROP TABLE CustCopy;
```

17 使用视图

17.1 为什么使用视图

1. 重用SQL语句
2. 简化复杂的SQL操作。在编写查询后，可以方便的重用而不必知道其细节。
3. 使用表的一部分而不是整个表。
4. 保护数据，可以授予用户访问表的特定部分的权限，而非整个表的访问权限。

17.2 创建视图

17.2.1 使用视图简化联结查询

```
1 CREATE VIEW ProductCustomers AS
2 SELECT cust_name, cust_contact, prod_id
3 FROM Customers, Orders, OrderItems
4 WHERE Customers.cust_id = Orders.cust_id
5 AND OrderItems.order_num = Orders.order_num;
```

[注]: 删除视图可以使用 `DROP VIEW viewname`

```
1 SELECT cust_name, cust_contact
2 FROM ProductCustomers
3 WHERE prod_id = 'RGAN01';
```

17.2.2 用视图重新格式化检索出的数据

```
1 CREATE VIEW VendorLocations AS
2 SELECT RTRIM(vend_name) + '(' + RTRIM(vend_country) +
3        ')'
4        AS vend_title
5 FROM Vendors;
```

```
1 SELECT * FROM VendorLocations;
```

18 高级SQL特性

18.1 约束

18.1.1 主键

1. 任意两行值不相同。
2. 每行都有一个主键值（即不允许 `NULL`）。
3. 包含主键值的列从不修改或更新。
4. 主键值不能重用，如果删掉某一行，其主键值不会重新分配给新行。

```
1 CREATE TABLE Vendors
2 (
3     vend_id CHAR(10) NOT NULL PRIMARY KEY,
4     vend_name CHAR(50) NOT NULL,
5     vend_address CHAR(50) NULL
6 );
```

18.1.2 外键

外键是表中的一列，其值必须列在另一表中的主键中。

```
1 CREATE TABLE Orders
2 (
3     order_num INTEGER NOT NULL PRIMARY KEY,
4     order_date DATETIME NOT NULL,
5     cust_id CHAR(10) NOT NULL REFERENCES
    Customers(cust_id)
6 );
```

[分析]: 其中的表定义使用了 `REFERENCES` 关键字，它表示 `cust_id` 中的任何值都必须是 `Customers` 表的 `cust_id` 值。

18.1.3 唯一约束

唯一约束用来保证一列中的数据是唯一的。它们类似主键，但是有以下区别：

1. 唯一约束列可以包含 `NULL` 值。
2. 唯一约束列可以修改或更新。
3. 唯一约束列的值可重复使用。
4. 唯一约束列不能用来定义外键。

使用 `UNIQUE` 约束

18.1.4 检查约束

```
1 CREATE TABLE OrderItems
2 (
3     order_num INTEGER NOT NULL,
4     order_item INTEGER NOT NULL,
5     prod_id CHAR(10) NOT NULL,
6     quantity INTEGER NOT NULL CHECK (quantity > 0),
7     item_price MONEY NOT NULL
8 );
```