



SQL

◎ type	Post
◎ status	Draft
📅 date	@2026/01/05
≡ summary	MySQL+copilot
≡ tags	工具
◎ category	SQL
≡ 标签	
※ 状态	进行中

1. 基本数据结构

表 – 结构化的文件，存储特定类型的数据。如：清单、目录等。

1. 每个表中数据的类型都相同。
2. 每个表都有独属于自己的唯一名字来表示自身。

3. 表的一些特性定义了数据在表中的存储方式，包含怎样的数据，数据如何分解等信息。

列



表中的一个字段，所有表都是由一个或多个竖向的列组成的。

每个表列都有相应的数据类型，它限制（或允许）该列中存储的数据。

行



表中的一个记录

表中的数据是按行存储的，所保存的每个记录存储在自己的行内。如果将表想象为网格，网格中垂直的列为表列，水平行为表行。

主键



一列（或一组列），其值能够唯一标识表中每一行。

表中的每一（几）行/列都可以唯一标识自己。而唯一标识表中每行的**这个列称为 主键**。用它来表示一个特定的行。没有主键，更新或者删除表中特定行极为困难。

列可充当主键的条件：

- 任意两行都不具有相同的主键值；
- 每一行都必须具有一个主键值（主键列不允许 NULL 值）；
- 主键列中的值不允许修改或更新；
- 主键值不能重用（如果某行从表中删除，它的主键不能赋给以后的新行）。

2. 数据检索 - SELECT

必须指明选择的对象以及从何处选择。

基本检索

```
SELECT name1, name2, FROM produccts;  
    对象1 对象2      对象范围
```

检索所有列

```
SELECT * FROM Products
```



一般而言，尽量不使用 * 通配符。因为检索不需要的列会降低检索和应用
程序的性能。

检索不同值 – 使用DISTINCT关键字。

```
SELECT DISTINCT id FROM Products;
```

尽管在id所标识的列中会有多个相同值的行，现在只会输出不同id的行。

限制结果

当我们想只返回某一数量的行时，SQL并不支'0'。

MySQL

```
SELECT prod_name  
FROM Products  
LIMIT 5;
```

在SQL server 和 ACCESS 中：

```
-- 只返回前五行  
SELECT TOP 5 prod_name  
FROM Products;
```

DB2

```
SELECT prod_name  
FROM Products
```

```
FETCH FIRST 5 ROWS ONLY;
```

Oracle

```
SELECT prod_name  
FROM Products  
WHERE ROWNUM <=5
```

注释方法

1. — 这是一条注释
2. # 这是一条注释
3. /*这是
一段注释*/

3. 排序检索数据 - ORDER BY

排序



子句SQL语句由子句构成，有些子句可选有些必须。一个子句通常由一个关键字加上所提供的数据组成。如上面提到的FROM就是 SELECT 的子句。在指定一条 ORDER BY 子句时，应该保证它是 SELECT 语句中最后一条子句。如果它不是最后的子句，将会出现错误消息。

```
SELECT prod_name  
FROM Products  
ORDER BY prod_name; -- 默认以prod_name 的字母顺序对数据排列
```

按多个列排序

简单指定列名，在ORDER BY 后的两个列名用逗号分开即可。

```
SELECT prod_id, prod_price, prod_name  
FROM Products  
ORDER BY prod_price, prod_name;
```

-- 输出

prod_id	prod_price	prod_name
BNBG02	3.4900	Bird bean bag toy
BNBG01	3.4900	Fish bean bag toy
BNBG03	3.4900	Rabbit bean bag toy
RGAN01	4.9900	Raggedy Ann
BR03	11.9900	18 inch teddy bear

按列位置排序

ORDER BY 还支持按相对列位置进行排序。

若我们希望先按照SELECT清单中的第二个列排序然后按照第三个列排序：

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

指定排序方向

降序： DESC

升序： ASC

```
SELECT prod_id, prod_price, prod_name  
FROM Products  
ORDER BY prod_price DESC, prod_name;
```

-- 按prod_price降序排序，并在其值相同时按照prod_name的字母顺序排序

4. 过滤 - WHERE



通常只会根据特定操作或报告的需要提取表数据的子集。只检索所需数据需要指定

定搜索条件 (search criteria)，搜索条件也称为过滤条件 (filter condition)。

数据根据 WHERE 子句中指定的搜索条件进行过滤。

-- 从 products 表中检索两个列，但不返回所有行，只返回 prod_price 值为 3.49 的行

```
SELECT prod_name, prod_price  
FROM Products  
WHERE prod_price = 3.49;
```



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 子句操作符

表 4-1 WHERE 子句操作符

操作符	说明
=	等于
<>	不等于
!=	不等于
<	小于
<=	小于等于
!<	不小于
>	大于
>=	大于等于
!>	不大于
BETWEEN	在指定的两个值之间
IS NULL	为 NULL 值



`!=` 和 `<>` 通常可以互换。但是，并非所有 DBMS 都支持这两种不等于操作符。

-- 检索所有价格小于等于 10 美元的产品

```
SELECT prod_name, prod_price  
FROM Products  
WHERE prod_price <= 10;
```

不匹配检查 - `!=` 和 `<>`

-- 列出所有不是供应商 DLL01 制造的产品

```
SELECT vend_id, prod_name  
FROM Products  
WHERE vend_id <> 'DLL01';
```



单引号用来限定字符串。如果将值与字符串类型的列进行比较，就需要限定引号。

用来与数值列进行比较的值不用引号。

范围值检查 - `BETWEEN`

它需要两个值，即范围的开始值和结束值。

-- 检索价格在 5 美元和 10 美元之间的所有产品。

```
SELECT prod_name, prod_price  
FROM Products  
WHERE prod_price BETWEEN 5 AND 10;
```

空值检查 - `IS NULL`



`NULL`

无值 (no value)，它与字段包含 0、空字符串或仅仅包含空格不同。

-- 返回所有没有价格（空 prod_price 字段，不是价格为 0）的产品

```
SELECT prod_name  
FROM Products  
WHERE prod_price IS NULL;
```

组合 WHERE 子句

AND 操作符 – 给 WHERE 子句附加条件

-- 检索由供应商 DLL01 制造且价格小于等于 4 美元的所有产品的名称和价格

```
SELECT prod_id, prod_price, prod_name  
FROM Products  
WHERE ven_id = 'DLL01' AND prod_price <= 4;
```

OR 操作符 - 检索匹配任一条件的行



第一个条件得到满足的情况下，就不再计算第二个条件

-- 检索由任一个指定供应商制造的所有产品的产品名和价格

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

求值顺序



SQL 在处理 OR 操作符前，优先处理 AND 操作符

-- 错误写法

-- 返回的行中有 4 行价格小于 10 美元，但是我们需要列出价格为 10 美元及以上。

```
SELECT prod_name, prod_price
```

-- 用括号提高 OR 的优先级

```
SELECT prod_name, prod_price  
FROM Products  
WHERE (vend_id = 'DLL01' OR ven
```

```

FROM Products
WHERE vend_id = 'DLL01' OR vend
_id = 'BRS01'
AND prod_price >= 10;
--输出
prod_name prod_price
-----
Fish bean bag toy 3.4900
Bird bean bag toy 3.4900
Rabbit bean bag toy 3.4900
18 inch teddy bear 11.9900
Raggedy Ann 4.9900

```

IN 操作符号 - 指定条件范围



范围中的每个条件都可以进行匹配。

其实 IN 和 OR 具有相同的功能，但是 IN 的最大优点在于可以包含其他 SELECT 语句，能动态地建立 WHERE 子句。

```

-- 检索由供应商 DLL01 和 BRS01 制
造的所有产品
SELECT prod_name, prod_price
FROM Products
WHERE vend_id IN ( 'DLL01', 'BRS0
1' )
ORDER BY prod_name;

```

```

-- same
SELECT prod_name, prod_price
FROM Products
WHERE vend_id = 'DLL01' OR vend
_id = 'BRS01'
ORDER BY prod_name;

```

NOT 操作符 - 否定其后所跟的任何条件。

```

SELECT prod_name
FROM Products
WHERE NOT vend_id = 'DLL01'
ORDER BY prod_name;

```

5 用通配符进行过滤



1、通配符 (wildcard)

用来匹配值的一部分的特殊字符。要在搜索子句中使用通配符，必须使用 `LIKE` 操作符。但是通配符只能用于文本字段（字符串），非文本数据类型字段不能使用通配符搜索。

2、搜索模式 (search pattern)

由字面值、通配符或两者构成的搜索条件。

谓词 - 当操作符作为谓词时就不再充当操作符

通配符 `%(in DBMS) * (in MA)`



`%` 表示任何字符出现任意次数。

-- 匹配任何位置上包含文本 bean bag 的值

```
SELECT prod_id, prod_name  
FROM Products  
WHERE prod_name LIKE '%bean bag%';
```

-- 找出所有以词 Fish 起头的产品

```
SELECT prod_id, prod_name  
FROM Products  
WHERE prod_name LIKE 'FIS  
H%';
```



`%` 不能与NULL匹配

通配符 下划线（_） 单个_只匹配单个字符



1. DB2 不支持 `_`
2. Access 中为 `?`

-- 搜索模式给出了后面跟有文本的两个通配符

```
SELECT prod_id, prod_name  
FROM Products  
WHERE prod_name LIKE '_ inch teddy bear';
```

通配符 ([]) - 必须匹配指定位置（通配符的位置）的一个字符

-- 找出所有名字以 J 或 M 起头的联系人

-- [JM] 匹配方括号中任意一个字符，它也只能匹配单个字符

-- % 通配符匹配第一个字符之后的任意数目的字符，返回所需结果

```
SELECT cust_contact  
FROM Customs  
WHERE cust_contact LIKE '[JM]%'  
ORDER BY cust_contact;
```



如果使用的是 Microsoft Access，需要用!而不是^来否定一个集合。