《SQL必知必会》学习笔记

Terminal start: mysql -h localhost -u root -p +enter

数据检索

关键字顺序：SELECT ... FROM ... WHERE ... GROUP BY...HAVING...ORDER BY ...

一般检索

```
SELECT prod_id,prod_name,prod_price FROM Products;
SELECT * FROM Products;
```

排序检索

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

等同于

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

默认升序排列，可DESC关键字降序排列：

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

过滤检索

where子句

```
SELECT prod_name,prod_price FROM Products WHERE prod_price=3.49;
SELECT prod_name,prod_price FROM Products WHERE prod_price BETWEEN 5 AND 10;
SELECT prod_name,prod_price FROM Products WHERE prod_price IS NULL;
```

where子句操作符

操作符	说明
>,=,!=,...	同一般逻辑运算符
BETWEEN	两值之间
IS NULL	为NULL值

组合where子句

AND/OR操作符

AND的优先级高于OR：

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

SELECT prod_id,prod_price,prod_name FROM Products
WHERE vend_id = 'DLL01' OR prod_price<= 4;

SELECT prod_price,prod_name FROM Products
WHERE  (vend_id = 'DLL01' OR vend_id='BRS01') AND prod_price <= 4;
```

IN 操作符

```
SELECT prod_name,prod_price FROM Products
WHERE vend_id IN ('DLL01','BRS01')
ORDER BY prod_name;
```

等同于

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

NOT 操作符[^1]

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

注意： MySQL中的NOT只用来否定EXISTS (如NOT EXISTS)

LIKE操作符(严格说不是操作符而是谓词predicate)

通配符	说明
%	任意，如 *
_	一个，如？
[]	集合，集合中的任一个
[^]	集合的否定，同NOT

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

SELECT prod_name FROM Products
WHERE prod_name LIKE 'F%y';

SELECT prod_name FROM Products
WHERE prod_name LIKE 'F%y%'; 比'F%y'更好,可避免值后面跟空格的情况。
```

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

```
SELECT cust_contact FROM Customers
WHERE cust_contact LIKE '[JM]%';

SELECT cust_contact FROM Customers
WHERE cust_contact LIKE '[^JM]%';
等同于
SELECT cust_contact FROM Customers
WHERE NOT cust_contact LIKE '[JM]%';
```

注意： MySQL不支持集合'[]'

字段拼接

一般操作

```
SELECT vend_name +' (' + vend_country + ')' FROM Vendors
ORDER BY vend_name;

SELECT vend_name || ' (' || vend_country || ')' FROM Vendors
ORDER BY vend_name;
```

去掉空格

上述返回值会按照列宽使用空格填充为文本值，去掉空格可使用`RTRIM ()`：

```
SELECT RTRIM(vend_name) + ' (' + RTRIM(vend_country) + ')'
FROM Vendors
ORDER BY vend_name;
```

函数	说明
RTRIM()	去掉右边空格
LTRIM()	去掉左边空格
TRIM()	去掉左右空格

注意： MySQL不支持使用`+`或`||`的拼接，而是使用`CONCAT`,同时自动去掉空格：

```
SELECT CONCAT(vend_name,' (',vend_country,')') FROM Vendors
ORDER BY vend_name;
```

注意： MySQL中`||`等同于操作符`OR`，`&&`等同于操作符`AND`

使用别名（也叫导出列derived column）

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

MySQL中：

```
SELECT CONCAT(vend_name,' (',vend_country,')')
AS vend_title FROM Vendors
ORDER BY vend_name;
```

字段计算

计算符：加减乘除`+` `-` `*` `/`

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

使用函数

函数差异

函数	语法
提取字符串	Access : MID() DB2、Oracle、PostgreSQL: SUBSTR() MySQL、SQL Server 、Sybase: SUBSTRING()
数据类型转换	Access、Oracle: 每种类型的转换都有一个函数 DB2、PostgreSQL: CAST() MySQL、SQL Server 、Sybase: CONVERT()
提取当前日期	Access： NOW() Oracle: SYSDATE DB2、PostgreSQL: CURRENT_DATE MySQL: CURDATE() SQL Server 、Sybase: GETDATE()

文本处理函数

函数	说明
LEFT()	返回串左边的字符
LENGTH()/DATALENGTH()/LEN()	返回串的长度
LOWER() [Access使用LCASE()]	小写
UPPER()[Access使用UCASE]	大写
LTRIM()/RTRIM()/TRIM()	去掉左/右/左右空格
SOUNDEX()	按语音表示

```
SELECT cust_name,cust_contact FROM Customers
WHERE SOUNDEX(cust_name) = SOUNDEX('Michael Green');
```

日期处理函数

函数	说明
DATEPART()	部分日期，适用SQL Server、Sybase和Access
DATE_PART()	适用PostgreSQL
YEAR()	MySQL
to_number(to_char(order_date,'YY'))	Oracle

```
SELECT order_num FROM Orders
WHERE DATEPART(yy,order_date) = 2004;
```

```
SELECT order_num FROM Orders
WHERE DATEPART('yyyy',order_date) = 2004;

SELECT order_num FROM Orders
WHERE DATE_PART('year',order_date) = 2004;

SELECT order_num FROM Orders
WHERE YEAR(order_date) = 2004;

SELECT order_num FROM Orders
WHERE to_number(to_char(order_date,'YY') = 2004;
```

数值处理函数

函数	说明
ABS()	绝对值
COS()	余弦
EXP()	指数值
PI()	圆周率
SIN()	正弦
SQRT()	平方根
TAN()	正切

聚集函数-返回标量

函数	说明
AVG()	平均值（跳过缺失值行）
COUNT()	行数（跳过缺失值行，但COUNT(*)不跳过）
MAX()	最大值（跳过缺失值行）
MIN()	最小值（跳过缺失值行）
SUM()	列和（跳过缺失值行）

```
SELECT AVG(DISTINCT prod_price) AS avg_price
FROM Products
WHERE vend_id ='DLL01'
```

分组检索

```
SELECT vend_id,COUNT(*) AS num_prods FROM Products
GROUP BY vend_id;
```

分组过滤

WHERE子句过滤的是特定行值，HAVING过滤是基于分组聚集值。若不指定GROUP BY则二者视为相同。

```
SELCET vend_id,COUNT(*) AS num_prods FROM Products
GROUP BY vend_id
HAVING COUNT(*)>=2;

SELECT order_num,COUNT(*) AS items FROM OrderItems
GROUP BY order_num
HAVING COUNT(*) >= 3
ORDER BY items,order_num;
```

子查询

利用子查询进行过滤

```
SELECT cust_name,cust_contact FROM Customers
WHERE cust_in IN (SELECT cust_id FROM Orders
                  WHERE order_num IN (
                  SELECT order_num FROM OrderItems
                  WHERE prod_id = 'RGAN01'));
```

利用子查询作为计算字段

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

联结查询

应该保证所有的联结都有WHERE子句，否则DBMS将返回比想要的多得多的数据

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

```
SELECT vend_name,prod_name,prod_price
FROM Vendors INNER JOIN Products
ON Vendors.vend_id = Products.vend_id;
```

```
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 = 20007;
```

高级联结

自联结

```
SELECT c1.cust_id,c1.cust_name,c1.cust_contact
FROM Customers AS c1,Customers AS c2
WHERE c1.cust_name = c2.cust_name
AND c2.cust_contact='Jim Jones';
```

自然联结

```
SELECT C.*,O.order_num,O.order_date,OI.prod_id,OI.quantity,OI.item_price
FROM Customers AS C,Orders AS O,OrderItems AS OI
WHERE C.cust_id = O.cust_id
AND OI.order_num = O.order_num
AND prod_id = 'RGAN01';
```

外部联结

内部联结

```
SELECT Customers.cust_id,Orders.order_num
FROM Customers INNER JOIN Orders
ON Customer.cust_id = Orders.cust_id;
```

左外部联结

```
SELECT Customers.cust_id,Orders.order_num
FROM Customers LEFT OUTER JOIN Orders
ON Customer.cust_id = Orders.cust_id;
```

等同于

```
SELECT Customers.cust_id,Orders.order_num
FROM Customers,Orders
WHERE Customer.cust_id *= Orders.cust_id;
```

右外部联结

```
SELECT Customers.cust_id,Orders.order_num
FROM Customers RIGHT OUTER JOIN Orders
ON Orders.cust_id = Customers.cust_id;
```

等同于

```
SELECT Customers.cust_id,Orders.order_num
FROM Customers,Orders
WHERE Customer.cust_id =* Orders.cust_id;
```

全外部联结

```
SELECT Customers.cust_id,Orders.order_num
FROM Customers FULL OUTER JOIN Orders
ON Orders.cust_id = Customers.cust_id;
```

注意： Access、MySQL、SQL Server 或Sybase不支持FULL OUTER JOIN 语法。

组合查询

```
SELECT cust_name,cust_contact,cust_email
FROM Customers
WHERE cust_state IN ('IL','IN','MI')
UNION
SELECT cust_name,cust_contact,cust_email
FROM Customers
WHERE cust_name = 'Fun4All';

SELECT cust_name,cust_contact,cust_email
FROM Customers
WHERE cust_state IN ('IL','IN','MI')
UNION ALL
SELECT cust_name,cust_contact,cust_email
FROM Customers
WHERE cust_name = 'Fun4All';
```

注意：

1. UNION操作会默认去掉重复的行，UNION ALL可保留重复的行。
2. 其他类型的UNION: EXCEPT(或称MINUS)、INTERSECT。

数据变动

插入数据

```
INSERT INTO Customers(
```

```
    cust_id,
    cust_name,
    cust_address,
    cust_city,
    cust_state,
    cust_zip,
    cust_country,
    cust_contact,
    cust_email)
VALUES(
    '10000006',
    'Toy Land',
    '123 Any Street',
    'New York',
    'NY',
    '11111',
    'USA',
    NULL,
    NULL
    );
```

```
INSERT INTO Customers(
    cust_id,
    cust_name,
    cust_address,
    cust_city,
    cust_state,
    cust_zip,
    cust_country,
    cust_contact,
    cust_email)
SELECT cust_id,
    cust_name,
    cust_address,
    cust_city,
    cust_state,
    cust_zip,
    cust_country,
    cust_contact,
    cust_email
FROM CustNew;
```

复制表

```
SELECT * INTO CustCopy
FROM Customers;
```

MySQL的语法不同：

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

更新数据

```
UPDATE Customers
SET cust_contact = 'Sam Roberts',
    cust_email = 'sam@toyland.com'
WHERE cust_id = '100000006';
```

删除数据(删除行)

```
DELETE FROM Customers
WHERE cust_id = '100000006';
```

若要删除所有行，不要适用DELETE。可使用速度更快的

```
TRUNCATE TABLE;
```

注意： 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。

创建和操纵表

创建表

```
CREATE TABEL OrderItems
 (
    order_num   INTEGER     NOT NULL,
    order_item  INTEGER     NOT NULL,
    prod_id     CHAR(10)    NOT NULL,
    quantity    INTEGER     NOT NULL    DEFAULT 1,
    item_price  DECIMAL(8,2) NOT NULL
);
```

注意： MySQL中`VARCHAR`要换为`TEXT`

更新表

添加列

```
ALTER TABLE Vendors
ADD vend_phone CHAR(20);
```

删除列

```
ALTER TABLE Vendors
DROP COLUMN vend_phone;
```

删除表

```
DROP TABLE CustCopy;
```

重命名表

命令	DBMS
RENAME oldname newname	DB2、MySQL、Oracle和PostgreSQL
sp_rename oldname newname	SQL Server 和Sybase

视图

视图包含的是一个查询，是虚拟的表，可以像使用表一样使用视图。

创建视图

```
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;
```

```
CREATE VIEW CustomerEMailList AS
SELECT cust_id,cust_name,cust_email
FROM Customers
WHERE cust_email IS NOT NULL;
```

存储过程

创建存储过程

Oracle版本：

```
CREATE PROCEDURE MailingListCount
(ListCount OUT NUMBER)
IS
BEGIN
    SELECT * FROM Customer
    WHERE NOT cust_email IS NULL;
    ListCount := SQL%ROWCOUNT;
END;
```

IN:传递值给存储过程
OUT:从存储过程返回值
INOUT：既传递值给存储过程也从存储过程返回值

SQL Server版本：

```
CREATE PROCEDURE MailingListCount
AS
DECLARE @cnt INTEGER
SELECT @cnt=COUNT(*)
FROM Customers
WHERE NOT cust_email IS NULL
RETURN @cnt;
```

SQL Server所有局部变量名都以@起头。

```
CREATE PROCEDURE NewOrder @cust_id CHAR(10)
AS
DECLARE @order_num INTEGER
SELECT @order_num = MAX(order_num)
FROM Orders
SELECT @order_num = @order_num+1
INSERT INTO Orders(order_num,order_date,cust_id)
VALUES(@order_num,GETDATE(),@cust_id)
RETURN @order_num;
```

```
CREATE PROCEDURE NewOrder @cust_id CHAR(10)
AS
INSERT INTO Orders(cust_id)
VALUES(@cust_id)
SELECT order_num = @@IDENTITY;
```

由DBMS生成订单号order_num，SQL Server称这些自动增量的列为标识字段（identity field），其他DBMS称为自动编号（auto number）或序列（sequence）。
@@IDENTITY为全局变量，此处指代自动生成的订单号。

执行存储过程

```
EXECUTE NewOrder(new_cust_id);
```

事务处理

开始事务与提交事务

SQL Server：

```
BEGIN TRANSACTION
DELETE OrderItems WHERE order_num = 12345
DELETE Orders WHERE order_num = 12345
COMMIT TRANSACTION
```

MySQL:

```
START TRANSACTION
DELETE OrderItems WHERE order_num = 12345
DELETE Orders WHERE order_num = 12345
COMMIT TRANSACTION
```

Oracle:

```
DELETE OrderItems WHERE order_num = 12345
DELETE Orders WHERE order_num = 12345
COMMIT;
```

回滚事务

```
DELETE FROM Orders;
ROLLBACK;
```

创建保留点和回退到保留点

MySQL和Oracle：

```
SAVEPOINT delete1;
ROLLBACK TO delete1;
```

SQL Server和Sybase：

```
SAVE TRANSACTION delete1;
ROLLBACK TRANSACTION delete1;
```

完整例子：

```
BEGIN TRANSACTION
INSERT INTO Customers(cust_id,cust_name)
VALUES('10000010','Toys Emporium');
SAVE TRANSACTION StartOrder;
INSERT INTO Orders(order_num,order_date,cust_id)
VALUES(20100,'2001/12/1','10000010');
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;
INSERT INTO OrderItems(order_num,order_item,prod_id,quantity,item_price)
VALUES(20010,1,'BR01',100,5.49);
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;
INSERT INTO OrderItems(order_num,order_item,prod_id,quantity,item_price)
VALUES(20010,2,'BR03',100,10.99);
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;
COMMIT TRANSACTION;
```

游标

创建游标

DB2、SQL Server和Sybase：

```
DECLARE CustCursor CURSOR
FOR
SELECT * FROM Customers
WHERE cust_email IS NULL
```

Oracle 和PostgreSQL：

```
DECLARE CURSOR CustCursor
IS
SELECT * FROM Customers
WHERE cust_email IS NULL
```

使用游标

```
OPEN CURSOR CustCursor
```

Oracle:

```
DECLARE TYPE CustCursor IS REF CURSOR
RETURN Customers%ROWTYPE;
DECLARE CustRecord Customers%ROWTYPE
BEGIN
    OPEN CustCursor;
    LOOP
    FETCH CustCursor INTO CustRecord;
    EXIT WHEN CustCursor%NOTFOUND;
    ...
    END LOOP;
    CLOSE CustCursor;
END;
```

SQL Server:

```
DECLARE @cust_id CHAR(10),
    @cust_name CHAR(50),
    @cust_address CHAR(50),
    @cust_city CHAR(50),
    @cust_state CHAR(5),
    @cust_zip CHAR(10),
    @cust_country CHAR(50),
    @cust_contact CHAR(50),
    @cust_email CHAR(255),
OPEN CustCursor
FETCH NEXT FROM CustCursor
    INTO @cust_id,@cust_name,@cust_address,
    @cust_city,@cust_state,@cust_zip,
    @cust_country,@cust_contact,@cust_email
WHILE @@FETCH_STATUS=0
BEGIN
...
FETCH NEXT FROM CustCursor
    INTO @cust_id,@cust_name,@cust_address,
    @cust_city,@cust_state,@cust_zip,
    @cust_country,@cust_contact,@cust_email
END
CLOSE CustCursor
```

关闭游标

DB2、Oracle和PostgreSQL:

```
CLOSE CustCursor
```

SQL Server:

```
CLOSE CustCursor
DEALLOCATE CURSOR CustCursor
```

高级SQL特性

主键

```
CREATE TABLE Vendors
(vend_id CHAR(10) NOT NULL PRIMARY KEY,
...
)
```

同样的功能实现

```
ALTER TABLE Vendors
ADD CONSTRAINT PRIMARY KEY (vend_id);
```

外键

```
CREATE TABLE Orders
(order_num INTEGER NOT NULL PRIMARY KEY,
 cust_id CHAR(10) NOT NULL REFERENCES Customers(cust_id)
);
```

同样的功能实现

```
ALTER TABLE Customers
ADD CONSTRAINT FOREIGN KEY (cust_id) REFERENCES Customers(cust_id);
```

检查约束

```
CREATE TABLE OrderItems
(
    order_num INTEGER NOT NULL,
    order_item INTEGER NOT NULL,
    prod_id CHAR(10) NOT NULL,
    quantity INTEGER NOT NULL CHECK(quantity>0),
    item_price MONEY NOT NULL
);
```

```
ADD CONSTRAINT CHECK (gender LIKE '[MF]')
```

索引

```
CREATE INDEX prod_name_ind
ON PRODUCTS (prod_name);
```

触发器

特殊的存储过程

SQL Server：

```
CREATE TRIGGER customer_state
ON Customers
FOR INSERT,UPDATE
AS
UPDATE Customers
SET cust_state = Upper(cust_state)
WHERE Customers.cust_id = inserted.cust_id;
```

Oracle和PostgreSQL:

```
CREATE TRIGGER customer_state
AFTER INSERT OR UPDATE
FOR EACH ROW
BEGIN
UPDATE Customers
SET cust_state = Upper(cust_state)
WHERE Customers.cust_id = :OLD.cust_id
END;
```