SQL基本概念

什么是SQL?

SQL是Structured Query Language的缩写，意思是结构化查询语言，是一种在数据库管理系统（Relational Database Management System, RDBMS）中查询数据，或通过RDBMS对数据库中的数据进行更改的语言

常见的RDBMS有：

- Oracle Database：甲骨文公司的RDBMS
- SQL Server ：微软公司的RDBMS
- DB2：IBM 公司的RDBMS
- PostgreSQL：开源的RDBMS
- MySQL ：开源的RDBMS

注：不同RDBMS的SQL语言略有不同

执行原理

用户在客户端通过SQL语言，将需要的数据和对数据进行的操作的请求发送给RDBMS，RDBMS 根据该语句的内容返回所请求的数据，或者对存储在数据库中的数据进行更新。

根据对RDBMS 赋予的指令种类的不同，SQL 语句可以分为以下三类：

- DDL（Data Definition Language，数据定义语言）
    - CREATE： 创建数据库和表等对象
    - DROP： 删除数据库和表等对象
    - ALTER： 修改数据库和表等对象的结构
- DML（Data Manipulation Language，数据操纵语言）
    - SELECT：查询表中的数据
    - INSERT：向表中插入新数据
    - UPDATE：更新表中的数据
    - DELETE：删除表中的数据
- DCL（Data Control Language，数据控制语言）
    - COMMIT： 确认对数据库中的数据进行的变更
    - ROLLBACK： 取消对数据库中的数据进行的变更
    - GRANT： 赋予用户操作权限
    - REVOKE： 取消用户的操作权限

DDL（Data Definition Language，数据定义语言）

创建数据库(CREATE)

```
CREATE DATABASE shop;
```

创建表(CREATE)

```
CREATE TABLE Product
(product_id     CHAR(4)      NOT NULL,
 product_name   VARCHAR(100) NOT NULL,
 product_type   VARCHAR(32)  NOT NULL,
 sale_price     INTEGER      ,
 purchase_price INTEGER      ,
 regist_date    DATE         ,
 PRIMARY KEY (product_id));
```

每一列的数据类型（后述）是必须要指定的，数据类型包括：

- INTEGER 整数型
- NUMERIC（ 全体位数, 小数位数）
- CHAR 定长字符串
- VARCHAR 可变长字符串
- DATE 日期型

删除表(DROP)

```
DROP TABLE Product;
```

更新表(ALTER)

```
-- (ADD COLUMN)
ALTER TABLE Product ADD COLUMN product_name_pinyin VARCHAR(100);
-- (DROP COLUMN)
ALTER TABLE Product DROP COLUMN product_name_pinyin;
-- (RENAME)
RENAME TABLE Poduct to Product;
```

DML（Data Manipulation Language，数据操纵语言）

基本语句

数据查询(SELECT)

```
--
SELECT * FROM Product;

--
SELECT product_id, product_name, purchase_price
FROM Product;

-- (AS)
SELECT product_id AS id,
product_name AS name,
purchase_price AS “”
FROM Product;

--
SELECT '2009-02-24' AS date, product_id, product_name
FROM Product;

-- (DISTINCT)
SELECT DISTINCT product_type
FROM Product;
```

查询条件(WHERE)

```
SELECT product_name, product_type
  FROM Product;
 WHERE product_type = '';
```

算数运算符

- 加 +
- 减 -
- 乘 *
- 除 /

```
SELECT product_name, sale_price, sale_price * 2 AS "sale_price_x2" FROM Product;
```

比较运算符

- 等于 =
- 不等于 <>
- 大于 >
- 大于等于 >=
- 小于 <
- 小于等于 <=

```
SELECT product_name, product_type, regist_date
  FROM Product
 WHERE regist_date < '2009-09-27';

--
SELECT product_name, sale_price, purchase_price
  FROM Product
 WHERE sale_price - purchase_price >= 500;

-- NULL
SELECT product_name, purchase_price
  FROM Product
 WHERE purchase_price IS NULL;

SELECT product_name, purchase_price
  FROM Product
 WHERE purchase_price IS NOT NULL;
```

逻辑运算符

- NOT

（也就是sale_price<1000）

```
SELECT product_name, product_type, sale_price
FROM Product
WHERE NOT sale_price >= 1000;
```

- AND

  AND运算符在其两侧的查询条件都成立时整个查询条件才成立，其意思相当于“并且”。

```
SELECT product_name, purchase_price
  FROM Product
 WHERE product_type = ''
   AND sale_price >= 3000;
```

- OR

运算符在其两侧的查询条件有一个成立时整个查询条件都成立，其意思相当于“或者”。

```
SELECT product_name, purchase_price
  FROM Product
 WHERE product_type = ''
    OR sale_price >= 3000;
```

分组聚合(GROUP BY)

- 常用的五个聚合函数：

- COUNT： 计算表中的记录数（行数）
- SUM： 计算表中数值列中数据的合计值
- AVG： 计算表中数值列中数据的平均值
- MAX： 求出表中任意列中数据的最大值
- MIN： 求出表中任意列中数据的最小值

```
SELECT product_type, COUNT(*)
  FROM Product
 GROUP BY product_type;

-- NULL
SELECT COUNT(purchase_price)
  FROM Product;

--
SELECT COUNT(DISTINCT product_type)
  FROM Product;

-- SUM/AVGMAX/MIN
SELECT MAX(regist_date), MIN(regist_date)
  FROM Product;
GROUP BYWHERESELECT

FROM  WHERE  GROUP BY  SELECT

  SELECT purchase_price, COUNT(*)
    FROM Product
   WHERE product_type = ''
GROUP BY purchase_price;
(HAVING)
  SELECT product_type, COUNT(*)
    FROM Product
GROUP BY product_type
  HAVING COUNT(*) = 2;
```

数据排序(ORDER BY)

- **子句的书写顺序**

  SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY
- **子句的执行顺序：**

  FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

```
 SELECT product_id, product_name, sale_price, purchase_price
    FROM Product
ORDER BY sale_price;
```

- **升序(ASC)或降序(DESC)** 注意：默认升序

```
SELECT product_id, product_name, sale_price, purchase_price
    FROM Product
ORDER BY sale_price DESC;
```

数据插入(INSERT)

```
--
INSERT INTO Product (product_id, product_name, product_type, sale_price, purchase_price, regist_date) VALUES
('0001', 'T', '', 1000, 500, '2009-09-20');

--
START TRANSACTION;
INSERT INTO Product VALUES ('0001', 'T', '', 1000, 500, '2009-09-20');
INSERT INTO Product VALUES ('0002', '', '', 500, 320, '2009-09-11');
INSERT INTO Product VALUES ('0003', 'T', '', 4000, 2800, NULL);
INSERT INTO Product VALUES ('0004', '', '', 3000, 2800, '2009-09-20');
INSERT INTO Product VALUES ('0005', '', '', 6800, 5000, '2009-01-15');
INSERT INTO Product VALUES ('0006', '', '', 500, NULL, '2009-09-20');
INSERT INTO Product VALUES ('0007', '', '', 880, 790, '2008-04-28');
INSERT INTO Product VALUES ('0008', '', '', 100, NULL,'2009-11-11');
COMMIT;

--
INSERT INTO ProductCopy (product_id, product_name, product_typesale_price, purchase_price, regist_date)
SELECT product_id, product_name, product_type, sale_price, purchase_price, regist_date
  FROM Product;
```

数据删除(DELETE)

```
--
DELETE FROM Product;

-- DELETE
DELETE FROM Product
 WHERE sale_price >= 4000;
```

数据更新(UPDATE)

```
--
UPDATE Product
   SET regist_date = '2009-10-10';

-- UPDATE
UPDATE Product
   SET sale_price = sale_price * 10
 WHERE product_type = '';

--
UPDATE Product
   SET sale_price = sale_price * 10,
       purchase_price = purchase_price / 2
 WHERE product_type = '';
```

进阶语句

视图

注意：定义视图时不能使用ORDER BY子句

```
-- (CREATE VIEW)
CREATE VIEW ProductSum (product_type, cnt_product)
AS
  SELECT product_type, COUNT(*)
    FROM Product
GROUP BY product_type;

--
SELECT product_type, cnt_product
  FROM ProductSum;

-- (DROP VIEW)
DROP VIEW ProductSum;
```

子查询（一次性视图）

```
-- FROMSELECT
SELECT product_type, cnt_product
  FROM
  (  SELECT product_type, COUNT(*) AS cnt_product
       FROM Product
   GROUP BY product_type
  ) AS ProductSum;

-- WHERE
SELECT product_id, product_name, sale_price
  FROM Product
 WHERE sale_price > (SELECT AVG(sale_price)
                       FROM Product);

--
SELECT product_type, product_name, sale_price
  FROM Product AS P1
 WHERE sale_price > (SELECT AVG(sale_price)
                       FROM Product AS P2
                      WHERE P1.product_type = P2.product_type
                   GROUP BY product_type);
```

注意：

- 能够使用常数或者列名的地方，无论是SELECT 子句、GROUP BY 子句、HAVING 子句，还是ORDER BY 子句，几乎所有的地方都可以使用标量子查询。
- 这里起到关键作用的就是在子查询中添加的WHERE 子句的条件。该条件的意思就是，在同一商品种类中对各商品的销售单价和平均单价进行比较。

函数

函数大致可以分为以下几种。

1. 算术函数（用来进行数值计算的函数）
2. 字符串函数（用来进行字符串操作的函数）
3. 日期函数（用来进行日期操作的函数）
4. 转换函数（用来转换数据类型和值的函数）
5. 聚合函数（用来进行数据聚合的函数）

- 算数函数

  ABS（数值）—— 绝对值

  MOD（被除数，除数）—— 求余

  ROUND（对象数值，保留小数的位数）—— 四舍五入
- 字符串函数

  CONCAT（字符串1，字符串2，字符串3）—— 拼接

  LENGTH（字符串）—— 字符串长度

  LOWER（字符串）—— 小写

UPPER（字符串）—— 大写

REPLACE（对象字符串，替换前的字符串，替换后的字符串）—— 替换

SUBSTRING（对象字符串 FROM 截取的起始位置 FOR 截取的字符数）—— 截取

- 日期函数

CURRENT_DATE —— 当前日期

CURRENT_TIME —— 当前时间

CURRENT_TIMESTAMP —— 当前的日期和时间

EXTRACT（日期元素 FROM 日期）

- 转换函数

CAST（转换前的值 AS 想要转换的数据类型）—— 类型转换

COALESCE（数据1，数据2，数据3……）—— 将NULL转换为其他值

谓词

- LIKE
- BETWEEN
- IS NULL和IS NOT NULL
- EXIST和NOT EXIST
- IN和 NOT IN

注意：在使用IN 和NOT IN 时是无法选取出NULL 数据的。

CASE表达式

```
SELECT product_name,
       CASE WHEN product_type = ''
            THEN CONCAT('A:', product_type)
            WHEN product_type = ''
            THEN CONCAT('B:', product_type)
            WHEN product_type = ''
            THEN CONCAT('C:',product_type)
            ELSE NULL
       END AS abc_product_type
  FROM Product;
```

表的加减法

- 表的加法(UNION)

```
SELECT product_id, product_name
   FROM Product
UNION
SELECT product_id, product_name
  FROM Product2;
```

通过UNION 进行并集运算时可以使用任何形式的SELECT 语句，WHERE、GROUP BY、HAVING 等子句都可以使用，但是ORDER BY 只能在最后使用一次。

注意：UNION会删去两个表中的重复记录。如果想保留重复记录，可以在UNION后面加ALL

- 选取表中的公共部分(INTERSECT)

MySQL不支持INTERSECT

- 表的减法(EXCEPT)

MySQL不支持EXCEPT

以列为单位对表进行联结(JOIN)

- 内联结(INNER JOIN)

```
SELECT SP.shop_id, SP.shop_name, SP.product_id, P.product_name, P.sale_price
      FROM ShopProduct AS SP
INNER JOIN Product AS P
        ON SP.product_id = P.product_id;
```

像这样使用联结运算将满足相同规则的表联结起来时，WHERE、GROUP BY、HAVING、ORDER BY 等工具都可以正常使用.

- 外联结(OUTER JOIN)

```
SELECT SP.shop_id, SP.shop_name, SP.product_id, P.product_name, P.sale_price
           FROM ShopProduct AS SP
LEFT OUTER JOIN Product AS P
             ON SP.product_id = P.product_id;
```

- 三张以上的表的联结

```
SELECT SP.shop_id, SP.shop_name, SP.product_id, P.product_name,
           P.sale_price,IP.inventory_quantity
      FROM ShopProduct AS SP
INNER JOIN Product AS P
        ON SP.product_id = P.product_id
INNER JOIN InventoryProduct AS IP
        ON SP.product_id = IP.product_id
     WHERE IP.inventory_id = 'P001';
```

DCL（Data Control Language，数据控制语言）

创建事务(START TRANSACTION) - 提交处理(COMMIT)

```
START TRANSACTION;
    -- T1000
    UPDATE Product
       SET sale_price = sale_price - 1000
     WHERE product_name = 'T';
    -- T1000
    UPDATE Product
       SET sale_price = sale_price + 1000
     WHERE product_name = 'T';
COMMIT;
```

取消处理(ROLLBACK)

```
START TRANSACTION;
    -- T1000
    UPDATE Product
       SET sale_price = sale_price - 1000
     WHERE product_name = 'T';
    -- T1000
    UPDATE Product
       SET sale_price = sale_price + 1000
     WHERE product_name = 'T';
ROLLBACK;
```