

第三章：聚合与排序

• 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

• 数据表Product

商品编号	商品名称	商品种类	销售单价	进货单价	登记日期
0001	T恤衫	衣服	1000	500	2009-09-20
0002	打孔器	办公用品	500	320	2009-09-11
0003	运动T恤	衣服	4000	2800	
0004	菜刀	厨房用具	3000	2800	2009-09-20
0005	高压锅	厨房用具	6800	5000	2009-01-15
0006	叉子	厨房用具	500		2009-09-20
0007	擦菜板	厨房用具	880	790	2008-04-28
0008	圆珠笔	办公用品	100		2009-11-11

• 计算销售单价和进货单价的合计值

```
SELECT
    SUM(sale_price), SUM(purchase_price)
FROM
    Product;
/*COUNT函数的结果根据参数的不同而不同。COUNT(*)会得到包含NULL的数据行数，
    而COUNT(<列名>)会得到NULL之外的数据行数。
    聚合函数会将NULL排除在外。但COUNT(*)例外，并不会排除NULL。
*/
```



- 使用聚合函数删除重复值（关键字DISTINCT）

```
SELECT
    SUM(sale_price), SUM(DISTINCT sale_price)
FROM
    Product;
-- 在聚合函数的参数中使用DISTINCT，可以删除重复数据。
```



对表进行分组 GROUP BY

- 按照商品种类统计数据行数

```
SELECT
    product_type, COUNT(*)
FROM
    Product
GROUP BY product_type;
-- 1.SELECT → 2.FROM → 3.WHERE → 4.GROUP BY
```

product_type	count
衣服	2
办公用品	2
厨房用具	4

聚合键中包含NULL的情况

```
SELECT
    purchase_price, COUNT(*)
FROM
    Product
GROUP BY purchase_price;
-- 聚合键中包含NULL时，在结果中会以“不确定”行（空行）的形式表现出来。
```

purchase_price	count
	2
320	1
500	1
5000	1
2800	2
790	1

聚合键为NULL的结果

使用WHERE子句时GROUP BY的执行结果

```
SELECT
    purchase_price, COUNT(*)
FROM
    Product
WHERE
    product_type = '衣服'
GROUP BY purchase_price;
-- 执行顺序: FROM → WHERE → GROUP BY → SELECT
```

purchase_price	count
500	1
2800	1

• GROUP BY常见误区

- 把聚合键之外的列名书写在SELECT子句之中(MySQL支持)
- 在GROUP BY子句中写了列的别名(PostgreSQL和MySQL支持)
- GROUP BY子句结果的显示是无序的。

为聚合结果指定条件 HAVING

- 从按照商品种类进行分组后的结果中，取出“包含的数据行数为2行”的组

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

-- 书写顺序 SELECT → FROM → WHERE → GROUP BY → HAVING

-- HAVING里只能是：• 常数 • 聚合函数 • GROUP BY子句中指定的列名（即聚合键）

product_type	count
衣服	2
办公用品	2

• HAVING 与 WHERE

-- 将条件书写在HAVING子句中的情况

```
SELECT
    product_type, COUNT(*)
FROM
```

```
Product
GROUP BY product_type
HAVING product_type = '衣服';
```

-- 将条件书写在WHERE子句中的情况

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

-- 聚合键所对应的条件不应该书写在HAVING子句当中，而应该书写在WHERE子句当中。

product_type	count
衣服	2

对查询结果进行排序 ORDER BY

- 按照销售单价由高到低（降序）进行排列

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

-- 未指定ORDER BY子句中排列顺序时会默认使用升序(ASC)进行排列。

-- 在ORDER BY子句中可以使用SELECT子句中定义的别名

-- 在ORDER BY子句中可以使用SELECT子句中未使用的列和聚合函数。

product_id	product_name	sale_price	purchase_price
0005	高压锅	6800	5000
0003	运动T恤	4000	2800
0004	菜刀	3000	2800
0001	T恤衫	1000	500
0007	擦菜板	880	790
0002	打孔器	500	320
0006	叉子	500	
0008	圆珠笔	100	

• 指定多个排序键

-- 按照销售单价和商品编号的升序进行排序

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

product_id	product_name	sale_price	purchase_price
0008	圆珠笔	100	
0002	打孔器	500	320
0006	叉子	500	
0007	擦菜板	880	790
0001	T恤衫	1000	500
0004	菜刀	3000	2800
0003	运动T恤	4000	2800
0005	高压锅	6800	5000

价格相同时按照商品编号的升序排列

• NULL的顺序

-- 按照进货单价的升序进行排列

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

-- 排序键中包含NULL时，会在开头或末尾进行汇总。

product_id	product_name	sale_price	purchase_price
0002	打孔器	500	320
0001	T恤衫	1000	500
0007	擦菜板	880	790
0003	运动T恤	4000	2800
0004	菜刀	3000	2800
0005	高压锅	6800	5000
0006	叉子	500	
0008	圆珠笔	100	

NULL会汇集
在开头或者
末尾

第四章： 数据更新

数据的插入（INSERT语句）

• INSERT语句

```
INSERT INTO <表名> (列1, 列2, 列3, ..... ) VALUES (值1, 值2, 值3, .....);
-- 列清单(列1, 列2, 列3, ..... )可以省略
-- 插入失败指的是希望通过 INSERT 语句插入的数据无法正常插入到表中，但之前已经插入的数据并不会被破坏
```

• 插入NULL

```
INSERT INTO ProductIns (product_id, product_name, product_type,
                        sale_price, purchase_price, regist_date)
VALUES ('0006', '叉子', '厨房用具', 500, NULL, '2009-09-20');
-- 插入 NULL 的列一定不能设置 NOT NULL 约束
```

• 通过显式方法插入默认值

```
INSERT INTO ProductIns (product_id, product_name, product_type,
                        sale_price, purchase_price, regist_date)
VALUES ('0007', '擦菜板', '厨房用具', DEFAULT, 790, '2009-04-28');
-- 在创建表的 CREATE TABLE 语句中设置 DEFAULT 约束来设定默认值
```

• 从其他表中复制数据

```
-- 将商品表中的数据复制到商品复制表中
INSERT INTO ProductCopy (product_id, product_name, product_type,
                        sale_price, purchase_price, regist_date)
SELECT product_id, product_name, product_type,
       sale_price, purchase_price, regist_date
FROM Product;
-- INSERT语句的SELECT语句中，可以使用WHERE子句或者GROUP BY子句等任何SQL语法
-- （但使用ORDER BY子句并不会产生任何效果）
```

数据的删除（DELETE语句）

- DROP TABLE 语句可以将表完全删除
- DELETE 语句会留下表（容器），而删除表中的全部数据

• 清空Product表

```
DELETE FROM Product;
-- DELETE语句的删除对象并不是表或者列，而是记录（行）。
```

• 指定删除对象的DELETE语句（搜索型DELETE）

```
-- 删除销售单价（sale_price）大于等于4000日元的数据
DELETE FROM Product
WHERE
    sale_price >= 4000;
-- 可以通过WHERE子句指定对象条件来删除部分数据。
-- DELETE 语句中不能使用 GROUP BY、HAVING和ORDER BY三类子句，而只能使用WHERE子句
```

product_id	product_name	product_type	sale_price	purchase_price	regist_date
0001	T恤衫	衣服	1000	500	2009-09-20
0002	打孔器	办公用品	500	320	2009-09-11
0004	菜刀	厨房用具	3000	2800	2009-09-20
0006	叉子	厨房用具	500		2009-09-20
0007	擦菜板	厨房用具	880	790	2008-04-28
0008	圆珠笔	办公用品	100		2009-11-11

• 只能删除表中全部数据的TRUNCATE语句

```
TRUNCATE Product;
```


数据的更新（UPDATE语句）

- 改变表中数据的UPDATE语句

```
UPDATE <表名>
    SET <列名> = <表达式>;
-- 将登记日期全部更新为“2009-10-10”
UPDATE Product
SET
    regist_date = '2009-10-10';
```

product_id	product_name	product_type	sale_price	purchase_price	regist_date
0001	T恤衫	衣服	1000	500	2009-10-10
0002	打孔器	办公用品	500	320	2009-10-10
0004	菜刀	厨房用具	3000	2800	2009-10-10
0006	叉子	厨房用具	500		2009-10-10
0007	擦菜板	厨房用具	880	790	2009-10-10
0008	圆珠笔	办公用品	100		2009-10-10

所有行的数据都被更新为“2009-10-10”

- 指定条件的UPDATE语句（搜索型UPDATE）

```
UPDATE <表名>
    SET <列名> = <表达式>
    WHERE <条件>;

-- 将商品种类为厨房用具的记录的销售单价更新为原来的10倍
UPDATE Product
SET
    sale_price = sale_price * 10
WHERE
    product_type = '厨房用具';
```

product_id	product_name	product_type	sale_price	purchase_price	regist_date
0001	T恤衫	衣服	1000	500	2009-10-10
0002	打孔器	办公用品	500	320	2009-10-10
0004	菜刀	厨房用具	30000	2800	2009-10-10
0006	叉子	厨房用具	5000		2009-10-10
0007	擦菜板	厨房用具	8800	790	2009-10-10
0008	圆珠笔	办公用品	100		2009-10-10

仅厨房用具的价格更新为原来的10倍了

• 使用NULL进行更新

```
-- 将商品编号为0008的数据（圆珠笔）的登记日期更新为NULL
UPDATE Product
SET
    regist_date = NULL
WHERE
    product_id = '0008';
-- 使用UPDATE语句可以将值清空为NULL（但只限于未设置NOT NULL约束的列）。
```

product_id	product_name	product_type	sale_price	purchase_price	regist_date
0001	T恤衫	衣服	1000	500	2009-10-10
0002	打孔器	办公用品	500	320	2009-10-10
0004	菜刀	厨房用具	30000	2800	2009-10-10
0006	叉子	厨房用具	5000		2009-10-10
0007	擦菜板	厨房用具	8800	790	2009-10-10
0008	圆珠笔	办公用品	100		

登记日期被更新为NULL

• 多列更新

```
-- 使用逗号对列进行分隔排列
UPDATE Product
SET
    sale_price = sale_price * 10,
    purchase_price = purchase_price / 2
WHERE
    product_type = '厨房用具';

-- 将列用()括起来的清单形式
UPDATE Product
SET
    (sale_price, purchase_price) = (sale_price * 10, purchase_price / 2)
```

```
WHERE
    product_type = '厨房用具';
```

product_id	product_name	product_type	sale_price	purchase_price	regist_date
0001	T恤衫	衣服	1000	500	2009-10-10
0002	打孔器	办公用品	500	320	2009-10-10
0004	菜刀	厨房用具	300000	1400	2009-10-10
0006	叉子	厨房用具	50000		2009-10-10
0007	擦菜板	厨房用具	88000	395	2009-10-10
0008	圆珠笔	办公用品	100		

厨房用具的销售单价更新为原来的10倍

厨房用具的进货单价更新为原来的一半

事务

- 事务是需要在同一个处理单元中执行的一系列更新处理的集合。

```
START TRANSACTION;
-- 将运动T恤的销售单价降低1000日元
UPDATE Product
SET
    sale_price = sale_price - 1000
WHERE
    product_name = '运动T恤';
-- 将T恤衫的销售单价上浮1000日元
UPDATE Product
SET
    sale_price = sale_price + 1000
WHERE
    product_name = 'T恤衫';
COMMIT;
```

- 事务的ACID特性

/*

原子性 (Atomicity)：原子性是指在事务结束时，其中所包含的更新处理要么全部执行，要么完全不执行，也就是要么占有一切要么一无所有。例如，在之前的例子中，在事务结束时，绝对不可能出现运动T恤的价格下降了，而T恤衫的价格却没有上涨的情况。该事务的结束状态，要么是两者都执行了 (COMMIT)，要么是两者都未执行 (ROLLBACK)。从事务中途停止的角度去考虑，就能比较容易理解原子性的重要性了。由于用户在一个事务中定义了两条 UPDATE 语句，DBMS 肯定不会只执行其中一条，否则就会对业务处理造成影响。

一致性 (Consistency)：一致性指的是事务中包含的处理要满足数据库提前设置的约束，如主键约束或者 NOT NULL 约束等。例如，设置了 NOT NULL 约束的列是不能更新为 NULL 的，试图插入违反主键约束的记录就会出错，无法执行。对事务来说，这些不合法的SQL 会被回滚。也就是说，这些 SQL 处理会被取消，不会执行。一致性也称为完整性。

隔离性 (Isolation)：隔离性指的是保证不同事务之间互不干扰的特性。该特性保证了事务之间不会互相嵌套。此外，在某个事务中进行的更改，在该事务结束之前，对其他事务而言是不可见的。因此，即使某个事务向表中添加了记录，在没有提交之前，其他事务也是看不到新添加的记录。

持久性 (Durability)：持久性也可以称为耐久性，指的是在事务（不论是提交还是回滚）结束后，DBMS 能够保证该时间点的数据状态会被保存的特性。即使由于系统故障导致数据丢失，数据库也一定能通过某种手段进行恢复。如果不能保证持久性，即使是正常提交结束的事务，一旦发生了系统故障，也会导致数据丢失，一切都需要从头再来。保证持久性的方法根据实现的不同而不同，其中最常见的就是将事务的执行记录保存到硬盘等存储介质中（该执行记录称为日志）。当发生故障时，可以通过日志恢复到故障发生前的状态。

*/