20180703

笔记基于《SQL 必知必会》第 4 版电子版。

第 1 课　了解 SQL

✧　1.1　数据库基础

① 数据库 (database)

数据库是一个以某种有组织的方式存储的数据集合(容器)。

人们通常用数据库来代表使用的数据库软件，这是不正确的。确切说，数据库软件称为数据库管理系统(DBMS)，数据库是通过 DBMS 创建和操纵的容器。

主流数据库管理系统(DBMS)：

```
· Apache Open Office Base；
· IBM DB2；
· Microsoft Access；
· Microsoft SQL Server（包括 Microsoft SQL Server Express）；
· MariaDB；
· MySQL；
· Oracle（包括 Oracle Express）；
· PostgreSQL；
· SQLite。
```

② 表 (table)

表是一种结构化的文件，可以存储某种特定类型的数据，存储在表中的数据是同一种类型的数据或清单。同一个数据库表名不能重复。

模式(schema)：关于数据库和表的布局及特性的信息。

描述数据在表中如何存储，存储什么样的数据等；描述数据库和其中表的关系。

③ 列和数据类型

列(column)：表中的一个字段，表由一个或多个列组成。

正确地将数据分解为多个列极为重要。

数据类型(datatype)定义列可以存储哪些数据。

注意：数据类型及其名称是 SQL 不兼容的一个主要原因。

大多数基本数据类型得到了一致的支持，但许多高级的数据类型却没有。更糟的是，偶然会有相同的数据类型在不同的 DBMS 中具有不同的名称。

④ 行(row)

行：表中的一行记录(record)。很多时候记录和行两个术语可以交替使用。

⑤ 主键(primary key)

主键：唯一标识表中每一行的一列(或一组列)。

保证每个表都有一个主键，可以方便数据库操作和管理。

表中任何列都可以作为主键，但需要满足要求：

1) 任意两行的主键值不同；
2) 每一行必须由主键值(主键列不允许 NULL)；
3) 主键列的值不允许修改或更新；
4) 主键值不能重用；如果删除了某行，其主键值不能赋给新行。

主键一般定义在一列上；也可以多个列作为主键，此时需要所有列值的组合必须唯一，但单个列值可以重复。

✧ 1.2 什么是 SQL

SQL(Structured Query Language)：结构化查询语言，是一门专门用来与数据库沟通的语言。几乎所有重要的 DBMS 都支持 SQL。

许多 DBMS 厂商通过增加语句或指令，对 SQL 进行了扩展。其目的是提供执行特定操作的额外功能或简化方法。虽然很有用，但一般都是针对各自的 DBMS。标准 SQL 由 ANSI 标准委员会管理，称为 ANSI SQL。所有主要的 DBMS，即使有自己的扩展，也都支持 ANSI SQL。

第 2 课 检索数据

SELECT 语句：从表中检索信息。

此处数据表使用书中提供的脚本创建。测试数据库使用 MySQL。
创建数据库 sqltest：

```
mysql> create database sqltest charset=utf8;
```

脚本两个文件 create.txt 和 populate.txt，分别表示创建表和插入数据，将 txt 文件改为 utf-8 编码，命令行执行：

```
$ mysql -u root -p dbname < create.txt
$ mysql -u root -p dbname < populate.txt
```

✧ 2.1 检索

```
SELECT column1, column2... FROM tablename; -- 查询指定列
SELECT * FROM tablename; -- 查询所有列
```

示例：

```
mysql> SELECT prod_id, prod_name, prod_price FROM products;
+---------+--------------------+------------+
| prod_id | prod_name          | prod_price |
+---------+--------------------+------------+
| BNBG01  | Fish bean bag toy  |       3.49 |
| BNBG02  | Bird bean bag toy  |       3.49 |
| BNBG03  | Rabbit bean bag toy|       3.49 |
| BR01    | 8 inch teddy bear  |       5.99 |
| BR02    | 12 inch teddy bear |       8.99 |
| BR03    | 18 inch teddy bear |      11.99 |
| RGAN01  | Raggedy Ann        |       4.99 |
| RYL01   | King doll          |       9.49 |
| RYL02   | Queen doll         |       9.49 |
+---------+--------------------+------------+
9 rows in set (0.06 sec)
```

注意：

1) 查询多个列之间使用逗号,分隔，但是最后一个列名不加逗号；
2) 多条 SQL 语句必须以分号;分隔。多数 DBMS 不需要在单条 SQL 语句后加分号，但也有 DBMS (如 MySQL)必须在单条 SQL 语句后加分号；
3) SQL 语句不区分大小写，但是许多开发人员喜欢将 SQL 关键字大写，列名和表名使用小写，这样代码易于阅读和调试；
4) 处理 SQL 语句时，所有空格都被忽略。SQL 语句可以写成一行，也可以分写在多行；
5) 慎用通配符*，虽然使用通配符可以省事，但是查询所有行性能会下降，一般在不知道列名时可以使用。

✧ 2.2 检索不同的值

检索 products 表所有产品供应商 id：

```
mysql> SELECT vend_id FROM products;
+---------+
| vend_id |
+---------+
| BRS01   |
| BRS01   |
| BRS01   |
| DLL01   |
| DLL01   |
| DLL01   |
| DLL01   |
| FNG01   |
| FNG01   |
+---------+
```

因为有 9 个商品，返回 9 行数据，但是只有 3 个产品供应商。

使用 **DISTINCT** 关键字可以只返回不同的值：

```
mysql> SELECT DISTINCT vend_id FROM products;
+---------+
| vend_id |
+---------+
| BRS01   |
| DLL01   |
| FNG01   |
+---------+
```

注意：

DISTINCT 关键字必须直接放在列名前面，而且作用于所有的列；
如果使用 **DISTINCT** 查询多列，只有多列内容完全相同才会合并。

✧ 2.3 限制结果

① SQL Server 和 Access 使用 TOP 关键字限制最多返回多少行：

```
SELECT TOP 5 prod_name FROM Products;
```

② DB2

```
SELECT prod_name FROM Products FETCH FIRST 5 ROWS ONLY;
```

③ Oracle，基于 ROWNUM(行计数器)来计算行

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

④ MySQL、MariaDB、PostgreSQL、SQLite，使用 LIMIT 子句：

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

MySQL 等使用 **LIMIT m OFFSET n** 表示跳过 n 行数据，查询后 m 行数据：

```
mysql> SELECT prod_id,prod_name FROM products LIMIT 5 OFFSET 5;
+---------+-------------------+
| prod_id | prod_name         |
+---------+-------------------+
| BR03    | 18 inch teddy bear |
| RGAN01  | Raggedy Ann       |
| RYL01   | King doll         |
| RYL02   | Queen doll        |
+---------+-------------------+
```

跳过 5 行从第 6 行开始返回 5 行数据，但是只有 9 行，返回 6,7,8,9 四行数据。

MySQL 和 MariaDB 支持简化版的 **LIMIT 3 OFFSET 4** 语句，即 **LIMIT 4,3**。

✧ 2.4 使用注释

随着 SQL 语句变长，复杂性增加，此时就需要添加注释，便于阅读。

```
-- 这是单行注释
/* 这是
多行注释 */
```

第 3 课 排序检索数据

SELECT 语句的 **ORDER BY** 子句

✧ 3.1 排序数据

SELECT 查询默认显示的顺序是插入到表中的顺序。使用 **ORDER BY** 子句可以将查询得到的数据按照某些列排序。

```
mysql> SELECT prod_id, prod_price, prod_name
    -> FROM products
    -> ORDER BY prod_price, prod_name;
+---------+------------+--------------------+
| prod_id | prod_price | prod_name          |
+---------+------------+--------------------+
| BNBG02  |       3.49 | Bird bean bag toy  |
| BNBG01  |       3.49 | Fish bean bag toy  |
| BNBG03  |       3.49 | Rabbit bean bag toy |
| RGAN01  |       4.99 | Raggedy Ann        |
| BR01    |       5.99 | 8 inch teddy bear  |
| BR02    |       8.99 | 12 inch teddy bear |
| RYL01   |       9.49 | King doll          |
| RYL02   |       9.49 | Queen doll         |
| BR03    |      11.99 | 18 inch teddy bear |
+---------+------------+--------------------+
```

注意：ORDER BY 子句必须是 **SELECT** 语句的最后一条子句。

按照多个列排序时，只有前面的列值相同时，才会按照后面的列进行排序。

✧ 3.4 指定排序方向

默认使用升序(**ASC**)排序；指定 **DESC** 关键字可以采用降序排序。

```
mysql> SELECT prod_id, prod_price, prod_name
    -> FROM products
    -> ORDER BY prod_price DESC, prod_name;
```

```
+---------+------------+--------------------+
| prod_id | prod_price | prod_name          |
+---------+------------+--------------------+
| BR03    |      11.99 | 18 inch teddy bear |
| RYL01   |       9.49 | King doll          |
| RYL02   |       9.49 | Queen doll         |
| BR02    |       8.99 | 12 inch teddy bear |
| BR01    |       5.99 | 8 inch teddy bear  |
| RGAN01  |       4.99 | Raggedy Ann        |
| BNBG02  |       3.49 | Bird bean bag toy  |
| BNBG01  |       3.49 | Fish bean bag toy  |
| BNBG03  |       3.49 | Rabbit bean bag toy|
+---------+------------+--------------------+
```

对 prod_price 指定 **DESC** 降序排序，prod_name 没有指定，使用默认的 **ASC**；
因为默认使用升序排序，所有 **ASC** 没什么卵用。

第 4 课　过滤数据

SELECT 语句的 **WHERE** 子句

✧ 4.1　使用 WHERE 子句

数据库表一般包含大量数据，很少需要检索表中的所有行。通常根据特定需要提取表数据的子集。只检索所需数据需要指定搜索条件(search criteria)，搜索条件也称为过滤条件(filter condition)。

在 **SELECT** 语句中，数据根据 **WHERE** 子句中指定的搜索条件进行过滤。

```
mysql> SELECT name, cv, groupname, grade
    -> FROM lovelive
    -> WHERE groupname = 'guilty kiss'
    -> ORDER BY grade DESC;
+----------+-----------+------------+-------+
| name     | cv        | groupname  | grade |
+----------+-----------+------------+-------+
| 小原鞠莉 | 鈴木愛奈  | Guilty Kiss | 3     |
| 桜内梨子 | 逢田梨香子 | Guilty Kiss | 2     |
| 津島善子 | 小林愛香  | Guilty Kiss | 1     |
+----------+-----------+------------+-------+
```

✧ 4.2　WHERE 子句操作符

=：等于；**<>**或**!=**：不等于；
<：小于；**<=**：小于等于；**!**：不小于；
>：大于；**>=**：大于等于；**!>**：不大于；
BETWEEN AND：在指定两个值之间；
IS NULL：为 NULL 值。

并非所有 DBMS 都支持这些操作符。

示例：范围值检查，价格位于 5~10 美元的产品

```
mysql> SELECT prod_name, prod_price
    -> FROM products
    -> WHERE prod_price BETWEEN 5 AND 10;
+--------------------+------------+
| prod_name          | prod_price |
+--------------------+------------+
```

```
| 8 inch teddy bear  |       5.99 |
| 12 inch teddy bear |       8.99 |
| King doll          |       9.49 |
| Queen doll         |       9.49 |
+--------------------+------------+
```

示例：空值检查

创建表时可以指定某列能否包含值，列不包含值时就是 **NULL**

```
mysql> SELECT cust_name
    -> FROM customers
    -> WHERE cust_email IS NULL;
+---------------+
| cust_name     |
+---------------+
| Kids Place    |
| The Toy Store |
+---------------+
```

第 5 课　高级数据过滤

✧　5.1　组合 WHERE 语句

使用逻辑操作符 **AND** 或 **OR**。

WHERE 子句可以有多个 **AND** 和 **OR** 构成高级过滤，**AND** 的优先级高于 **OR**。

```
mysql> SELECT name, groupname, height
    -> FROM lovelive
    -> WHERE groupname = 'azalea'
    -> OR grade = 1 AND height > 160;
+-----------------+-----------+--------+
| name            | groupname | height |
+-----------------+-----------+--------+
| 西木野真姫      | BiBi      |    161 |
| 松浦果南        | AZALEA    |    162 |
| 黒澤ダイヤ      | AZALEA    |    162 |
| 国木田花丸      | AZALEA    |    152 |
+-----------------+-----------+--------+
```

✧　5.2　IN 操作符

IN 指定条件范围，**IN** (value1, value2 ...)，对范围中每一个条件都进行匹配。

```
mysql> SELECT name, height
    -> FROM lovelive
    -> WHERE height IN (155, 159);
+--------------+--------+
| name         | height |
+--------------+--------+
| 南ことり     |    159 |
| 園田海未     |    159 |
| 星空凛       |    155 |
| 東條希       |    159 |
+--------------+--------+
```

IN 是 **WHERE** 子句用来指定要匹配值清单的关键字，功能和 **OR** 相当，colomn **IN** (value1, value2 ...) 相当于 column = value1 **OR** column = value2 **OR** ...

IN 操作符的优点：

1) **IN** 操作符更加直观，一般比一组 **OR** 操作符执行更快；

2. IN 最大的优点是可以包含其他 SELECT 语句，能动态的建立 WHERE 子句。

✧ 5.3 NOT 操作符

WHERE 子句 NOT 操作符只有一个功能：否定其后所跟的任何条件。

```
mysql> SELECT name, bloodType, bust
    -> FROM lovelive
    -> WHERE bloodType NOT IN ('A', 'B')
    -> AND bust > 82;
+----------------+-----------+------+
| name           | bloodType | bust |
+----------------+-----------+------+
| 東條希         | O         |   90 |
| 松浦果南       | O         |   83 |
| 国木田花丸     | O         |   83 |
| 小原鞠莉       | AB        |   87 |
+----------------+-----------+------+
```

对于简单的 WHERE 子句使用 NOT 没有什么优势；但是对于更复杂的子句 NOT 可以简单地找出与条件列表不匹配的行。

第 6 课　使用通配符进行过滤

✧ 6.1 LIKE 操作符

通配符(wildcard)：用来匹配值的一部分的特殊字符。

搜索模式(search pattern)：由字面值、通配符或两者组合构成的搜索条件。

LIKE 操作符表示模糊查询，指示 DBMS 后面的搜索模式使用通配符而不是简单的相等匹配进行比较。

严格来说，LIKE 不是操作符，而是谓词(predicate)，虽然最终结果一样。

① 百分号%通配符

在搜索串中，%表示任何字符出现任意次数，不会匹配 NULL 的行。

```
mysql> SELECT name FROM lovelive WHERE name LIKE '%果%';
+----------------+
| name           |
+----------------+
| 高坂穂乃果     |
| 松浦果南       |
+----------------+
```

如果使用 Microsoft Access，需要使用*而不是%

② 下划线_通配符

在搜索串中，下划线_只匹配单个字符，而且只能是一个字符。

DB2 不支持通配符_；Microsoft Access 使用?而不是_。

```
mysql> SELECT name, birthday FROM lovelive WHERE birthday LIKE '_月_日';
+----------------+----------+
| name           | birthday |
+----------------+----------+
| 高坂穂乃果     | 8月3日   |
| 東條希         | 6月9日   |
| 高海千歌       | 8月1日   |
| 黒澤ダイヤ     | 1月1日   |
| 国木田花丸     | 3月4日   |
+----------------+----------+
```

③ 方括号[]通配符
方括号通配符用来指定一个字符集，必须匹配指定位置的一个字符。只有微软的 Access 和 SQL Server 支持。
WHERE name LIKE '[JM]%'表示匹配 J 或 M 开头的名字。

✧ 6.2 使用通配符的技巧
1) 使用通配符一般耗时更长，不要过度使用通配符，其他操作符也能达到相同目的时不要用通配符；
2) 尽量不要把通配符放在搜索模式的开始；把通配符置于开始处，搜索起来最慢的。

20180705

第 7 课 创建计算字段

✧ 7.1 计算字段
存储在数据库的数据很有可能不是程序需要的，需要从数据库中检索出转换、计算或格式化过的数据。计算字段并不实际存在于数据库表中；计算字段是运行时在 SELECT 语句内创建的。

字段(field)：基本与列(column)的意思相同，经常互换使用；不过数据库列一般称为列，而术语字段通常与计算字段一起使用。

在 SQL 语句内可完成的许多转换和格式化操作都可以直接在客户端应用程序内完成，但一般在数据库服务器上比在客户端中完成要快得多。

✧ 7.2 拼接字段
拼接(concatenate)：将值联结到一起构成单个值。
也就是将表中几列数据拼接到一起。

① 拼接的操作符：
1) Access 和 SQL Server 使用+；
2) DB2、Oracle、PostgreSQL、SQLite 和 Open Office Base 使用||；
3) MySQL 和 MariaDB 必须使用特殊的函数 CONCAT()。

```
mysql> SELECT CONCAT(name, ' (', groupname, ')')
    -> FROM lovelive
    -> WHERE id BETWEEN 1 AND 9;
+--------------------------------+
| CONCAT(name,' (',groupname,')') |
+--------------------------------+
| 高坂穂乃果 (Printemps)          |
| 絢瀬絵里 (BiBi)                 |
| 南ことり (Printemps)            |
| 園田海未 (lily white)           |
| 星空凛 (lily white)             |
| 西木野真姫 (BiBi)               |
| 東條希 (lily white)             |
| 小泉花陽 (Printemps)            |
| 矢澤にこ (BiBi)                 |
+--------------------------------+
```

② TRIM 函数
大多数 DBMS 都支持 **RTRIM**()、**LTRIM**()和 **TRIM**()。
1) **RTRIM**()：去掉字符串右边空格；
2) **LTRIM**()：去掉字符串左边空格；
3) **TRIM**()：去掉字符串左边和右边空格。

③ 使用别名
拼接的字段没有名字，只是一个值；如果只是简单的 SQL 查询自然是无所谓，但是如果想用于客户端应用，该字段没有名字，客户端无法引用。

别名(alias)是一个字段或值的替换名，使用关键字 AS：

```
mysql> SELECT CONCAT(name, ' (', groupname, ')')
    -> AS info
    -> FROM lovelive
    -> WHERE id BETWEEN 1 AND 9;
```

使用 **AS** 指定别名 info，指示 SQL 创建一个包含指定计算结果的名为 info 的计算字段，任何客户端都可以按名称引用此列，就像实际存在的列一样。
别名有时也称为导出列(derived column)。

✧ 7.3 执行算术计算
计算字段的另一常见用途是对查询的数据进行算术计算。
SQL 算术运算符：**+-*/**

Orders 表包含收到的所有订单，OrderItems 表包含每个订单中的各项物品。
计算指定订单每项商品总共卖出多少钱：

```
mysql> SELECT prod_id,
    ->        quantity,
    ->        item_price,
    ->        quantity * item_price AS expand_price
    -> FROM orderitems
    -> WHERE order_num = 20008;
+---------+----------+------------+--------------+
| prod_id | quantity | item_price | expand_price |
+---------+----------+------------+--------------+
| RGAN01  |        5 |       4.99 |        24.95 |
| BR03    |        5 |      11.99 |        59.95 |
| BNBG01  |       10 |       3.49 |        34.90 |
| BNBG02  |       10 |       3.49 |        34.90 |
| BNBG03  |       10 |       3.49 |        34.90 |
+---------+----------+------------+--------------+
```

如何测试计算
SELECT 语句为测试、检验函数和计算提供了很好的方法。虽然 **SELECT** 通常用于从表中检索数据，但省略 **FROM** 子句就是简单地访问和处理表达式，例如：

```
mysql> SELECT 3 * 2;
+-------+
| 3 * 2 |
+-------+
|     6 |
+-------+
```

```
mysql> SELECT TRIM('    a bc    ');
+----------------------+
| TRIM('    a bc    ') |
+----------------------+
| a bc                 |
+----------------------+

mysql> SELECT NOW(); --返回当前日期和时间
+---------------------+
| NOW()               |
+---------------------+
| 2018-07-05 09:29:34 |
+---------------------+
```

第 8 课　使用数据处理函数

✧ 8.1　函数

SQL 支持函数处理数据，但每个 DBMS 都有特定的函数，很有可能功能相同，但是函数名和语法完全不同。

DBMS 函数的差异：

1) 提取字符串的组成部分(子串)
   - Access 使用 MID()；
   - DB2、Oracle、PostgreSQL 和 SQLite 使用 SUBSTR()；
   - MySQL 和 SQL Server 使用 SUBSTRING()
2) 数据类型转换
   - Access 和 Oracle 使用多个函数，每种类型的转换有一个函数；
   - DB2 和 PostgreSQL 使用 CAST()；
   - MariaDB、MySQL 和 SQL Server 使用 CONVERT()
3) 获取当前日期
   - Access 使用 NOW()；
   - DB2 和 PostgreSQL 使用 CURRENT_DATE；
   - MariaDB 和 MySQL 使用 CURDATE()；
   - Oracle 使用 SYSDATE；
   - SQL Server 使用 GETDATE()；
   - SQLite 使用 DATE()；

可移植性(portable)：编写的代码可以在第一个系统上运行。
大部分 SQL 语句是可移植的；但 SQL 函数不具备可移植性，所以许多 SQL 程序员不建议使用 SQL 函数。

✧ 8.2　使用函数

大部分 SQL 实现支持以下类型的函数

① 文本处理函数

函数	说明
LEFT(str, len) / **RIGHT**(str, len) / 或使用子字符串函数如 **SUBSTRING**(str, pos[, len])	返回字符串左边/右边的字符
LTRIM(str) / RTRIM(str) / TRIM(str)	去掉字符串左边/右边/左右两边的空格
LOWER(str) / **UPPER**(str) (Access 使用 LCASE() / UCASE())	将字符串转换为小写/大写
SOUNDEX(str)	返回字符串的 SOUNDEX 值
LENGTH(str)(也使用 DATALENGTH()或 LEN())	返回字符串长度

SOUNDEX 是一个将任何文本串转换为描述其语音表示的字母数字模式的算法。SOUNDEX 考虑了类似的发音字符和音节，使得能对字符串进行发音比较而不是字母比较。
虽然 SOUNDEX 不是 SQL 概念，但多数 DBMS 都提供对 SOUNDEX 的支持，Microsoft Access 和 PostgreSQL 不支持。

比如 Customers 表中有一个顾客 Kids Place，其联系名为 Michelle Green。但如果这是错误的输入，此联系名实际上应该是 Michael Green，该怎么办呢？
直接查询：

```
mysql> SELECT cust_name, cust_contact
    -> FROM customers
    -> WHERE cust_contact = 'Michael Green';
Empty set (0.08 sec)
```

使用 SOUNDEX()进行发音查询：

```
mysql> SELECT cust_name, cust_contact
    -> FROM customers
    -> WHERE SOUNDEX(cust_contact) = SOUNDEX('Michael Green');
+------------+----------------+
| cust_name  | cust_contact   |
+------------+----------------+
| Kids Place | Michelle Green |
+------------+----------------+
```

因为 Michael Green 和 Michelle Green 的 SOUNDEX 值相同，能正确过滤得到需要的数据。

② 日期和时间处理函数
非常遗憾：日期和时间处理函数在 SQL 实现中差别很大，可移植性非常差。

MySQL 可以使用 **YEAR()**、**MONTH()**、**DATE()**、**DAY()**、**HOUR()**、**MINUTE()**、**SECOND()** 等提取日期或时间的部分。

```
mysql> SELECT cust_id, order_num, order_date
    -> FROM orders
    -> WHERE MONTH(order_date) = 2;
+------------+-----------+---------------------+
| cust_id    | order_num | order_date          |
+------------+-----------+---------------------+
| 1000000005 |     20008 | 2012-02-03 00:00:00 |
| 1000000001 |     20009 | 2012-02-08 00:00:00 |
+------------+-----------+---------------------+
```

DATE_FORMAT()可以获取格式化日期或时间字符串

```
mysql> SELECT date_format('2018-07-05', '%Y年%m月%d日');
+----------------------------------------+
| date_format('2018-07-05','%Y年%m月%d日') |
+----------------------------------------+
| 2018年07月05日                          |
+----------------------------------------+
```

③ 数值处理函数
仅处理数值数据，一般主要用于代数、三角或几何运算，使用不如字符串和日期时间处理函数频繁。讽刺的是，数值处理函数在 SQL 的最一致的函数。
常用有：**ABS()**、**COS()**、**EXP()**、**PI()**、**SIN()**、**SQRT()**、**TAN()**等。

第 9 课　汇总数据

✧ 9.1 聚集函数

聚集函数(aggregate function)对某些运行的函数，计算并返回一个值。

SQL 提供了 5 个聚集函数：

① **AVG**()

AVG()只能用来确定特定数值列的平均值，列名必须作为函数参数。**AVG**()忽略列值为 **NULL** 的行。

```
mysql> SELECT AVG(prod_price) AS avg_price
    -> FROM products;
+-----------+
| avg_price |
+-----------+
|  6.823333 |
+-----------+
```

② **COUNT**()

用于计数。

1) **COUNT**(*)：对表中行数进行计数。无论是不是 **NULL**；

2) **COUNT**(column)：对表特定列具有值进行计数，忽略 **NULL** 值。

```
mysql> SELECT COUNT(*) AS cust_num
    -> FROM customers;
+----------+
| cust_num |
+----------+
|        5 |
+----------+

mysql> SELECT COUNT(cust_email) AS cust_num
    -> FROM customers;
+----------+
| cust_num |
+----------+
|        3 |
+----------+
```

③ MAX()

MAX()返回指定列中的最大值，一般用于数值或日期值。有些 DBMS 允许返回任意列的最大值，对于文本数据，MAX()返回该列排序后最后一行。

MAX()忽略列值为 NULL 的行。

④ MIN()

MIN()返回指定列的最小值，用法与 MAX()类似。

⑤ SUM()

返回指定列值的和(总计)。

```
mysql> SELECT SUM(quantity) AS items_ordered
    -> FROM orderitems
    -> WHERE order_num = 20005;
+---------------+
| items_ordered |
+---------------+
|           200 |
+---------------+

mysql> SELECT SUM(item_price * quantity) AS total_price
    -> FROM orderitems
```

```
    -> WHERE order_num = 20005;
+-------------+
| total_price |
+-------------+
|     1648.00 |
+-------------+
```

利用标准的算术运算符，所有的聚集函数都可以执行多个列上的计算。
SUM()函数忽略列值为 **NULL** 的行。

✧ 9.2 聚集不同的值

对所有行执行计数，不指定参数默认使用 **ALL**；
使用关键字 **DISTINCT** 过滤相同的值。

注意：
Microsoft Access 在聚集函数中不支持 **DISTINCT**，需要使用子查询把 **DISTINCT** 数据返回到外部 **SELECT COUNT**(*)语句。

查询指定供应商提供产品的平均价格(只考虑不同价格)：

```
mysql> SELECT prod_price
    -> FROM products
    -> WHERE vend_id = 'DLL01';
+------------+
| prod_price |
+------------+
|       3.49 |
|       3.49 |
|       3.49 |
|       4.99 |
+------------+

mysql> SELECT AVG(DISTINCT prod_price) AS avg_price
    -> FROM products
    -> WHERE vend_id = 'DLL01';
+-----------+
| avg_price |
+-----------+
|  4.240000 |
+-----------+
```

注意：
如果指定列名，则 **DISTINCT** 只能用于 **COUNT**()。**DISTINCT** 不能用于 **COUNT**(*)。类似地，**DISTINCT** 必须使用列名，不能用于计算或表达式。

✧ 9.3 组合聚集函数

SELECT 语句可以根据需求包含多个聚集函数。

```
mysql> SELECT COUNT(*) AS num_items,
    ->        MIN(prod_price) AS min_price,
    ->        MAX(prod_price) AS max_price,
    ->        SUM(prod_price) AS sum_price
    -> FROM products;
+-----------+-----------+-----------+-----------+
| num_items | min_price | max_price | sum_price |
+-----------+-----------+-----------+-----------+
|         9 |      3.49 |     11.99 |     61.41 |
+-----------+-----------+-----------+-----------+
```

第 10 课　分组数据

之前的计算都是在表的所有数据或匹配特定的 WHERE 子句的数据上进行的。
使用分组可以将数据分为多个逻辑组，对每个组进行聚集计算。

✧　10.1 创建分组

使用 **SELECT** 语句的 **GROUP BY** 子句。
统计每个供应商提供的产品数目：

```
mysql> SELECT vend_id, COUNT(*) AS num_prods
    -> FROM products
    -> GROUP BY vend_id;
+---------+-----------+
| vend_id | num_prods |
+---------+-----------+
| BRS01   |         3 |
| DLL01   |         4 |
| FNG01   |         2 |
+---------+-----------+
```

GROUP BY 指示 DBMS 分组数据，然后对每个组而不是整个结果集进行聚集。

GROUP BY 使用规定：

1) 可以包含任意数目的列，因而可以对分组进行嵌套，更细致的进行分组；
2) 如果嵌套了分组，数据将在最后指定的分组上进行汇总；
3) **GROUP BY** 子句中列出的每一列必须是检索列(不能是聚集函数)。如果在 **SELECT** 中使用表达式，则在 **GROUP BY** 子句中必须使用相同表达式，不能使用别名。
4) 大多数 SQL 不允许 **GROUP BY** 列带有长度可变的数据类型；
5) 除了聚集函数，**SELECT** 语句中每一列必须在 **GROUP BY** 子句中给出；
6) 如果分组列中包含具有 **NULL** 值的行，**NULL** 将作为一个分组返回；
7) **GROUP BY** 必须在 **WHERE** 之后，**ORDER BY** 之前。

```
mysql> SELECT groupname, grade, COUNT(*)
    -> FROM lovelive
    -> WHERE id BETWEEN 1 AND 9
    -> GROUP BY groupname, grade;
+------------+-------+----------+
| groupname  | grade | count(*) |
+------------+-------+----------+
| BiBi       | 1     |        1 |
| BiBi       | 3     |        2 |
| lily white | 1     |        1 |
| lily white | 2     |        1 |
| lily white | 3     |        1 |
| Printemps  | 1     |        1 |
| Printemps  | 2     |        2 |
+------------+-------+----------+
```

✧　10.2　过滤分组

WHERE 可以过滤指定的行，但是不能用于分组。
对于分组过滤使用 **HAVING**，用法与 **WHERE** 类似。

```
mysql> SELECT groupname, grade, COUNT(*)
    -> FROM lovelive
    -> WHERE id BETWEEN 1 AND 9
    -> GROUP BY groupname, grade
    -> HAVING COUNT(*) > 1;
+-----------+-------+----------+
| groupname | grade | count(*) |
+-----------+-------+----------+
| BiBi      | 3     |        2 |
| Printemps | 2     |        2 |
+-----------+-------+----------+
```

Where 和 **Having** 的区别：
1) **Where** 过滤的是行，在分组之前对原始数据筛选；
2) **Having** 过滤的是分组，对分组后的数据筛选；

✧ 10.3 分组和排序

ORDER BY	**GROUP BY**
对产生的输出排序	对行分组，但输出可能不是分组的顺序
任意列都可以使用	只能使用选择列或表达式列，而且必须使用每个选择列表达式
不一定需要	如果与聚集函数一起使用列(或表达式)，则必须使用

一般使用 **GROUP BY** 子句后，也应该使用 **ORDER BY** 子句，这是保证数据正确排序的唯一方法。

```
mysql> SELECT groupname, grade, COUNT(*)
    -> FROM lovelive
    -> WHERE id BETWEEN 1 AND 9
    -> GROUP BY groupname, grade
    -> ORDER BY COUNT(*), groupname;
+------------+-------+----------+
| groupname  | grade | count(*) |
+------------+-------+----------+
| BiBi       | 1     |        1 |
| lily white | 1     |        1 |
| lily white | 2     |        1 |
| lily white | 3     |        1 |
| Printemps  | 1     |        1 |
| BiBi       | 3     |        2 |
| Printemps  | 2     |        2 |
+------------+-------+----------+
```

✧ 10.4 SELECT 子句顺序

子句	说明	是否必须使用
SELECT	要返回的列或表达式	是
FROM	查询数据的表	仅在从表选择数据时
WHERE	行级过滤	否
GROUP BY	分组	仅在按组计算聚集时
HAVING	组级过滤	否
ORDER BY	输出排序顺序	否

第 11 课　子查询

✧　11.1　子查询

查询(query)：任何 SQL 语句都是查询，但此术语一般指 SELECT 语句。
SQL 还可以创建子查询(subquery)，即嵌套在其他查询中的查询。

✧　11.2 利用子查询进行过滤

假如需要列出订购物品 RGAN01 的所有顾客，应该怎样检索?

1) 从订单详情表 OrderItems 中查询包含订购物品 RGAN01 的所有订单编号；

```
mysql> SELECT order_num
    -> FROM orderitems
    -> WHERE prod_id = 'rgan01';
+-----------+
| order_num |
+-----------+
|     20007 |
|     20008 |
+-----------+
```

2) 根据订单编号，从订单表 Orders 中查询顾客 ID；

```
mysql> SELECT cust_id
    -> FROM orders
    -> WHERE order_num IN (20007, 20008);
+------------+
| cust_id    |
+------------+
| 1000000004 |
| 1000000005 |
+------------+
```

3) 根据顾客 ID，从顾客表 Customers 查询顾客信息(如 cust_name)。

```
mysql> SELECT cust_name
    -> FROM customers
    -> WHERE cust_id IN ('1000000004', '1000000005');
+---------------+
| cust_name     |
+---------------+
| Fun4All       |
| The Toy Store |
+---------------+
```

结合 3 个查询，将前面的查询变为子查询：

```
mysql> SELECT cust_name
    -> FROM customers
    -> WHERE cust_id IN (SELECT cust_id
    ->                   FROM orders
    ->                   WHERE order_num IN (SELECT order_num
    ->                                       FROM orderitems
    ->                                       WHERE prod_id = 'rgan01'));
+---------------+
| cust_name     |
+---------------+
| Fun4All       |
| The Toy Store |
+---------------+
```

实际执行 3 条 SELECT 语句。里面子查询返回订单号列表，用于外层子查询的 WHERE 子句；外层子查询返回顾客 ID 列表，用于最外层查询的 WHERE 子句；最外层查询返回需要的数据。

注意：

1) 包含子查询的 SELECT 语句难以阅读和调试，把子查询分解为多行并进行适当的缩进，能极大地简化子查询的使用；
2) 好的 DBMS 客户端使用颜色代码 SQL 也可以方便阅读；
3) 子查询使用灵活，但由于性能限制，不要嵌套太多的子查询；
4) 作为子查询的 SELECT 语句只能查询单个列。查询多个列返回错误。

✧ 11.3 作为计算字段使用子查询

假如需要显示 Customers 表中每个顾客的订单总数，该怎样写？

1) 从 Customers 表中检索顾客列表。

```
mysql> SELECT cust_name, cust_state
    -> FROM customers
    -> ORDER BY cust_name;
+---------------+------------+
| cust_name     | cust_state |
+---------------+------------+
| Fun4All       | IN         |
| Fun4All       | AZ         |
| Kids Place    | OH         |
| The Toy Store | IL         |
| Village Toys  | MI         |
+---------------+------------+
```

2) 对于检索的每个顾客，统计在 Orders 表中的数目。

针对其中 id 为 1000000001 的用户统计：

```
mysql> SELECT COUNT(*) AS orders
    -> FROM orders
    -> WHERE cust_id = '1000000001';
+--------+
| orders |
+--------+
|      2 |
+--------+
```

使用子查询：

```
mysql> SELECT cust_name,
    ->        cust_state,
    ->        (SELECT COUNT(*)
    ->         FROM orders
    ->         WHERE orders.cust_id = customers.cust_id) AS orders
    -> FROM customers
    -> ORDER BY cust_name;
+---------------+------------+--------+
| cust_name     | cust_state | orders |
+---------------+------------+--------+
| Fun4All       | IN         |      1 |
| Fun4All       | AZ         |      1 |
| Kids Place    | OH         |      0 |
| The Toy Store | IL         |      1 |
| Village Toys  | MI         |      2 |
+---------------+------------+--------+
```

此处使用了完全限定列名 orders.cust_id 和 customers.cust_id，而不只是列名 cust_id，因为 orders 表和 customers 表都有 cust_id 列，如果不明确是哪个表的 cust_id，就容易混淆，产生问题。

如：SELECT COUNT(*) FROM orders WHERE cust_id = cust_id

DBMS 就会认为 orders 表的 cust_id 与自己比较，返回 orders 表订单总数。这不

是期望的结果。

注意：
WHERE 和 ORDER BY 子句指定的列可能出现在多个表中，如果在 SELECT 语句操作多个表，应该使用完全限定列名避免歧义。

第 12 课　联结表

✧ 12.1　联结

SQL 最强大的功能之一就是能在数据查询的执行中联结(join)表。联结是利用 SQL 的 SELECT 能执行的最重要的操作。

关系表的设计就是把信息分成多个表，一类数据一个表。各表通过某些共同的值进行关联，所以叫关系数据库。
比如供应商表 vendors 和产品表 products：vendors 表每一行是一个供应商，具有唯一标识就是主键 ID；products 除了存储产品信息，还需要存储对应供应商 ID (vendors 表的主键，也叫 products 的外键)，这样两个表就关联起来。
外键(foreign key)：外键为某个表的一列，包含另一个表的主键值，定义了两个表之间的关系。

这样的好处是：

1) 供应商信息不重复，不会浪费时间和空间；
2) 如果供应商信息变动，只需更新 vendors 表的记录，相关表中的数据不用改动；
3) 由于数据不重复，数据显然是一致的，使得处理数据和生成报表更简单。

可伸缩(scale)：能够适应不断增加的工作量而不失败。设计良好的数据库或应用程序称为可伸缩性好(scale well)。

联结是一种机制，不是物理实体，在实际数据库中并不存在。DBMS 根据需要建立联结，在查询执行期间一直存在。

✧ 12.2　创建联结

当 FROM 子句后接多个表时候，表示联结产生了。SELECT 子句后面所接的字段，分别来自于两个表中。

```
mysql> select prod_name, prod_price, vend_name, vend_city
    -> from products, vendors
    -> where vendors.vend_id = products.vend_id
    -> order by prod_price;
+--------------------+------------+----------------+------------+
| prod_name          | prod_price | vend_name      | vend_city  |
+--------------------+------------+----------------+------------+
| Fish bean bag toy  |       3.49 | Doll House Inc. | Dollsville |
| Bird bean bag toy  |       3.49 | Doll House Inc. | Dollsville |
| Rabbit bean bag toy |       3.49 | Doll House Inc. | Dollsville |
| Raggedy Ann        |       4.99 | Doll House Inc. | Dollsville |
| 8 inch teddy bear  |       5.99 | Bears R Us      | Bear Town  |
| 12 inch teddy bear |       8.99 | Bears R Us      | Bear Town  |
| King doll          |       9.49 | Fun and Games   | London     |
| Queen doll         |       9.49 | Fun and Games   | London     |
| 18 inch teddy bear |      11.99 | Bears R Us      | Bear Town  |
+--------------------+------------+----------------+------------+
```

① WHERE 子句的重要性

数据库表的定义没有指示 DBMS 如何对表进行联结，必须开发者做这件事。在联结两个表时，实际要做的是将第一个表中的每一行与第二个表中的每一行配对。WHERE 子句作为过滤条件，返回只包含匹配给定条件的行。

没有 WHERE 子句，第一个表中的每一行将与第二个表中的每一行配对，而不管它们逻辑上是否能配在一起，返回结果是笛卡尔积。

笛卡尔积(cartesian product)：A×B={(x,y)|x∈A∧y∈B}

如 A={a,b}, B={0,1,2}，则 A×B={(a, 0), (a, 1), (a, 2), (b, 0), (b, 1), (b, 2)}

也就是联结查询没有 WHERE 子句，返回行数是第一个表的行数乘以第二个表的行数。

注意：

保证所以联结都有 WHERE 子句，否则返回的数据比需要的多；当然也要保证 WHERE 子句的正确性。

有时返回笛卡尔积的联结也称为叉联结(cross join)。

② 内联结

目前使用的联结是等值联结(equijoin)，基于两个表之间的相等测试，也叫内联结(inner join)。对此种联结也可以明确指定联结的类型。

```
mysql> select prod_name, prod_price, vend_name, vend_city
    -> from vendors
    -> inner join products on vendors.vend_id = products.vend_id
    -> order by prod_price;
+--------------------+------------+----------------+------------+
| prod_name          | prod_price | vend_name      | vend_city  |
+--------------------+------------+----------------+------------+
| Fish bean bag toy  |       3.49 | Doll House Inc. | Dollsville |
| Bird bean bag toy  |       3.49 | Doll House Inc. | Dollsville |
| Rabbit bean bag toy |       3.49 | Doll House Inc. | Dollsville |
| Raggedy Ann        |       4.99 | Doll House Inc. | Dollsville |
| 8 inch teddy bear  |       5.99 | Bears R Us      | Bear Town  |
| 12 inch teddy bear |       8.99 | Bears R Us      | Bear Town  |
| King doll          |       9.49 | Fun and Games   | London     |
| Queen doll         |       9.49 | Fun and Games   | London     |
| 18 inch teddy bear |      11.99 | Bears R Us      | Bear Town  |
+--------------------+------------+----------------+------------+
```

两个表之间关系是以 INNER JOIN 指定的部分 FROM 子句，此时使用 ON 子句给出联结条件。传递给 ON 的实际条件与传递给 WHERE 的相同。

③ 联结多个表

查询订单 20007 的产品名、产品价格、数量、供应商名：

```
mysql> select prod_name, prod_price, quantity, vend_name
    -> from orderItems, products, vendors
    -> where products.vend_id = vendors.vend_id
    -> and orderItems.prod_id = products.prod_id
    -> and order_num = 20007;
+--------------------+------------+----------+----------------+
| prod_name          | prod_price | quantity | vend_name      |
+--------------------+------------+----------+----------------+
| 18 inch teddy bear |      11.99 |       50 | Bears R Us     |
| Fish bean bag toy  |       3.49 |      100 | Doll House Inc. |
| Bird bean bag toy  |       3.49 |      100 | Doll House Inc. |
| Rabbit bean bag toy |       3.49 |      100 | Doll House Inc. |
| Raggedy Ann        |       4.99 |       50 | Doll House Inc. |
+--------------------+------------+----------+----------------+
```

注意：
联结可能非常耗费资源，所以不要联结不必要的表，联结表越多，性能越差。

第 11 课的例子：列出订购物品 RGAN01 的所有顾客，使用子查询实现；也可以使用联结查询实现。

```
mysql> select cust_name, cust_contact
    -> from customers, orders, orderitems
    -> where customers.cust_id = orders.cust_id
    -> and orders.order_num = orderitems.order_num
    -> and prod_id='rgan01';
+---------------+--------------------+
| cust_name     | cust_contact       |
+---------------+--------------------+
| Fun4All       | Denise L. Stephens |
| The Toy Store | Kim Howard         |
+---------------+--------------------+
```

20180706

第 13 课 创建高级联结

✧ 13.1 使用表别名
使用表别名优点：
1) 缩短 SQL 语句；
2) 允许在一条 select 语句中多次使用相同的表。

将上一个例子使用别名：

```
mysql> select cust_name, cust_contact
    -> from customers as c, orders as o, orderitems as oi
    -> where c.cust_id=o.cust_id
    -> and o.order_num=oi.order_num
    -> and prod_id='rgan01';
+---------------+--------------------+
| cust_name     | cust_contact       |
+---------------+--------------------+
| Fun4All       | Denise L. Stephens |
| The Toy Store | Kim Howard         |
+---------------+--------------------+
```

表别名不仅可以用于 WHERE 子句，也可以用于 SELECT 的列表、ORDER BY 子句和其他语句部分。

注意：
1) Oracle 没有 AS 关键字，所以相当于 AS 被省略了：customers c
2) 表别名只在查询执行时使用，与列别名不一样，表别名不会返回客户端。

✧ 13.2 使用不同类型的联结
四种联结：
1) 内联结或等值联结；
2) 自联结(self-join)；
3) 自然联结(natural join)；
4) 外联结(outer join)。

① 自联结

查询 lovelive 表中和星空凛同一小组的成员：
1) 使用子查询

```
mysql> select id, name, groupname
    -> from lovelive
    -> where groupname = (select groupname
    ->                    from lovelive
    ->                    where name = '星空凛');
+----+----------+------------+
| id | name     | groupname  |
+----+----------+------------+
|  4 | 園田海未 | lily white |
|  5 | 星空凛   | lily white |
|  7 | 東條希   | lily white |
+----+----------+------------+
```

2. 使用自联结

```
mysql> select l1.id,l1.name,l1.groupname from lovelive as l1,lovelive as l2
    -> where l1.groupname=l2.groupname and l2.name='星空凛';
+----+----------+------------+
| id | name     | groupname  |
+----+----------+------------+
|  4 | 園田海未 | lily white |
|  5 | 星空凛   | lily white |
|  7 | 東條希   | lily white |
+----+----------+------------+
```

建议使用自联结而不用子查询。自联结通常作为外部语句，用来替代从相同表中检索数据的子查询语句，因为许多 DBMS 处理联结远比子查询快得多。

② 自然联结
标准的联结(前一章的内部联结)返回所有数据，甚至相同的列多次出现。自然联结使每个列只返回一次。
自然联结开发者只能选择那些唯一的列，一般通过对表使用通配符(SELECT *)，对所有其他表的列使用明确的子集来完成的。

```
mysql> 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 o.order_num=oi.order_num
    -> and prod_id='rgan01';
```

此例通配符只对第一个表使用。所有其他列明确列出，所以没有重复的列被检索出来。事实上，迄今为止建立的每个内联结都是自然联结，基本上用到的内联结都是自然联结。

③ 外联结
许多联结将一个表中的行和另一个表中的行相关联，但有时候需要包含没有关联行的那些行。如：
1) 对每个顾客订单进行计数，包括至今尚未下订单的顾客。
2) 列出所有产品以及订购数量，包括没人订购的产品。

联结包含了那些相关表中没有关联的行称为外联结。

例如：查询所有顾客 id 和其订单编号：
1) 内联结

```
mysql> SELECT customers.cust_id, orders.order_num
    -> FROM customers INNER JOIN orders
    ->  ON customers.cust_id = orders.cust_id;
+------------+-----------+
| cust_id    | order_num |
+------------+-----------+
| 1000000001 |     20005 |
| 1000000001 |     20009 |
| 1000000003 |     20006 |
| 1000000004 |     20007 |
| 1000000005 |     20008 |
+------------+-----------+
```

没有下订单的顾客不会出现在内联结表中。

2) 外联结

```
mysql> SELECT customers.cust_id, orders.order_num
    -> FROM customers LEFT OUTER JOIN orders
    ->  ON customers.cust_id = orders.cust_id;
+------------+-----------+
| cust_id    | order_num |
+------------+-----------+
| 1000000001 |     20005 |
| 1000000001 |     20009 |
| 1000000002 |      NULL |
| 1000000003 |     20006 |
| 1000000004 |     20007 |
| 1000000005 |     20008 |
+------------+-----------+
```

此处使用关键字 OUTER JOIN 指定联结类型，必须使用 LEFT 或 RIGHT 关键字指定包括其所有行的表(LEFT 是 OUTER JOIN 左边的表)。
此例使用 LEFT OUTER JOIN 从左边的表 customers 选择所有行。

注意：左外联结和右外联结唯一差别是所关联表的顺序；调整 FROM 或 WHERE 子句表的顺序，两者可以相互转换。

全外联结(full outer join)：查询两个表所有行并关联那些可以关联的行。但是 Access、MariaDB、MySQL、SQLite 等不支持 FULL OUTER JOIN 语法。

✧ 13.3 使用带聚集函数的联结
如检索所有顾客及每个顾客所下的订单数：

```
mysql> select customers.cust_id, count(orders.order_num) as num_ord
    -> from customers
    -> inner join orders on customers.cust_id=orders.cust_id
    -> group by customers.cust_id;
+------------+---------+
| cust_id    | num_ord |
+------------+---------+
| 1000000001 |       2 |
| 1000000003 |       1 |
| 1000000004 |       1 |
| 1000000005 |       1 |
+------------+---------+
```

内联结忽略了没有订单的顾客，如果要显示所有顾客，需要左外联结：

```
mysql> select customers.cust_id, count(orders.order_num) as num_ord
    -> from customers
    -> left outer join orders on customers.cust_id=orders.cust_id
    -> group by customers.cust_id;
+------------+---------+
| cust_id    | num_ord |
+------------+---------+
```

```
| 1000000001 |       2 |
| 1000000002 |       0 |
| 1000000003 |       1 |
| 1000000004 |       1 |
| 1000000005 |       1 |
+------------+---------+
```

LEFT OUTER JOIN 可以简写为 LEFT JOIN；RIGHT 同理。

✧ 13.4 使用联结和联结条件

联结及其使用要点：

1) 注意使用的联结类型，有内联结，外联结等；
2) 联结语法每个 DBMS 可能不一样；
3) 保证使用正确的联结条件，否则返回不正确的数据；
4) 应该总是使用联结条件，否则会得到笛卡尔积；
5) 在一个联结中可以包含多个表，甚至可以对每个联结采用不同的联结类型。这样做合法而且有用，但是应该先分别测试每个联结，再测试整体效果。

第 14 课 组合查询

使用 UNION 操作符将多条 SELECT 语句合成一个结果集。

✧ 14.1 组合查询

SQL 允许执行多条查询语句(多条 SELECT 语句)，并将结果作为一个查询结果集返回。组合查询通常称为并(union)或复合查询(compound query)

需要使用组合查询的情况：

1) 在一个查询中从不同的表返回结构数据；
2) 对一个表执行多个查询，按一个查询返回数据。

注意：

多数情况，组合相同表的两个查询所完成的工作与具有多个 WHERE 子句的一个查询所完成的工作相同。也就是，任何具有多个 WHERE 子句的 SELECT 语句都可以作为一个组合查询。

✧ 14.2 创建组合查询

用操作符 UNION 操作符组合多条 SELECT 语句，将结果组合成一个结果集。

① 使用 UNION

如查询 lovelive 表 bust>82 和 lily white 小组成员信息。

1) 使用 WHERE ... OR ...

```
mysql> alter table lovelive change column `groupname` `group` varchar(64);
Query OK, 0 rows affected (1.44 sec)

mysql> select id, name, bust, waist, hip
from lovelive
where bust > 82 or `group` = 'lily white';
+----+------------+------+-------+------+
| id | name       | bust | waist | hip  |
+----+------------+------+-------+------+
|  2 | 絢瀬絵里   |   88 |    60 |   84 |
|  4 | 園田海未   |   76 |    58 |   80 |
|  5 | 星空凛     |   75 |    59 |   80 |
|  7 | 東條希     |   90 |    60 |   82 |
```

```
| 12 | 松浦果南   |   83 |    58 |   84 |
| 16 | 国木田花丸 |   83 |    57 |   83 |
| 17 | 小原鞠莉   |   87 |    60 |   84 |
+----+------------+------+-------+------+
```

2) 使用 UNION

```
mysql> select id, name, bust, waist, hip
    -> from lovelive
    -> where bust > 82
    -> union
    -> select id, name, bust, waist, hip
    -> from lovelive
    -> where `group` = 'lily white';
+----+------------+------+-------+------+
| id | name       | bust | waist | hip  |
+----+------------+------+-------+------+
|  2 | 絢瀬絵里   |   88 |    60 |   84 |
|  7 | 東條希     |   90 |    60 |   82 |
| 12 | 松浦果南   |   83 |    58 |   84 |
| 16 | 国木田花丸 |   83 |    57 |   83 |
| 17 | 小原鞠莉   |   87 |    60 |   84 |
|  4 | 園田海未   |   76 |    58 |   80 |
|  5 | 星空凛     |   75 |    59 |   80 |
+----+------------+------+-------+------+
```

Union 和 Where 子句比较：

1) 对于简单的例子，使用 UNION 可能更复杂；对于较复杂的过滤条件，或者从多个表检索数据的情形，使用 UNION 可能会使处理更简单。

2) 多数 DBMS 使用内部查询优化程序，使用 Union 处理多条 SELECT 语句前会在内部组合它们，所以理论上性能几乎无差别。

② UNION 规则

1) UNION 必须由两条或以上的 SELECT 语句组成，语句之间用关键字 UNION 分隔；

2) UNION 每个查询必须包含相同的列、表达式、聚集函数(不需要以相同的次序列出)；

3) 列数据类型必须兼容：类型不必完全相同，但必须是 DBMS 可以隐含转换的类型。

③ 包含或取消重复的行

UNION 默认从查询结果集中自动去除了重复的行。如果想返回所有匹配的行，使用 UNION ALL，不会去除重复的行。

④ 对组合查询结果排序

用 UNION 组合查询时，只能使用一条 ORDER BY 子句，必须位于最后一条 SELECT 语句之后。

```
mysql> select id, name, bust, waist, hip
    -> from lovelive
    -> where bust > 82
    -> union
    -> select id, name, bust, waist, hip
    -> from lovelive
    -> where `group` = 'lily white'
    -> order by bust desc, hip desc, waist;
+----+------------+------+-------+------+
| id | name       | bust | waist | hip  |
+----+------------+------+-------+------+
|  7 | 東條希     |   90 |    60 |   82 |
|  2 | 絢瀬絵里   |   88 |    60 |   84 |
| 17 | 小原鞠莉   |   87 |    60 |   84 |
| 12 | 松浦果南   |   83 |    58 |   84 |
```

```
| 16 | 国木田花丸 |   83 |    57 |   83 |
|  4 | 園田海未   |   76 |    58 |   80 |
|  5 | 星空凛     |   75 |    59 |   80 |
+----+------------+------+-------+------+
```

虽然 ORDER BY 子句似乎只是最后一条 SELECT 语句的组成部分，但实际上 DBMS 用它排序所有 SELECT 语句返回的所有结果。

第 15 课　插入数据

INSERT 语句将数据插入表中。

✧　15.1　数据插入

插入的 3 种方式：插入完整的行；插入行的一部分；插入某些查询的结果。
使用 INSERT 语句可能需要客户端/服务器 DBMS 中的特定安全权限。

① 插入完整的行

1) 指定插入的表名+插入到新行的值

```
mysql> INSERT INTO `lovelive` VALUES
    -> (19, '上原步夢', '3月1日', '2', '大西亜玖璃', NULL, 82, 58, 84, 159, 'A');
Query OK, 1 row affected (0.12 sec)
```

存储到每一列的数据在 VALUES 子句给出，必须给每列提供一个值。如果某列没有值，使用 NULL(需要允许为空值)。

某些 SQL 实现中，INTO 关键字是可选的，但是为了保证可移植性，最好使用 INTO 关键字。

这种语法很简单，但是不安全，应该尽量避免使用。因为上面 SQL 语句高度依赖于表中列的定义顺序。

2) 指定插入的表名+列名+插入到新行的值

```
mysql> INSERT INTO `lovelive`(id, name, birthday, grade, cv, `group`,
    ->                        bust, waist, hip, height, bloodType)
    -> VALUES(20, '中須かすみ', '1月23日', '1', '相良茉優', NULL, 76, 55, 79, 155, 'B');
Query OK, 1 row affected (0.11 sec)
```

在表名后的括号明确给出列名，DBMS 将 VALUES 列表相应位置的值插入对应列；因为提供了列名，VALUES 必须以其指定的次序匹配指定的列名，不一定按各列出现在表中的实际次序。

注意：
1) 不要使用没有明确给出列的 INSERT 语句;给出列使 SQL 代码继续发挥作用，即使表结构发生变化；
2) VALUES 的数目必须正确，否则产生错误，数据插入失败。

② 插入部分行

使用 INSERT 的推荐方法明确给出列名，还可以省略列，只提供部分列的值。
省略列的前提是：
1) 改了允许为 NULL 值；
2) 表中定义该列的默认值。

③ 插入检索出的数据

INSERT 还可以将 SELECT 语句的结果插入表中，这就是所谓的 INSERT SELECT，它是由一条 INSERT 语句和一条 SELECT 语句组成的。

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

从 custnew 表查询数据插入到 customers 表中。

注意：

1) INSERT 和 SELECT 不一定要求列名一致，它使用的是列的位置；

2) INSERT 通常只插入一行。但 INSERT SELECT 是个例外，可以用一条 INSERT 插入多行，不管 SELECT 语句返回对少条语句，都会被插入到表中。

✧ 15.2 从一个表复制到另一个表

SELECT INTO 语句将数据复制到一个全新的表中。

DB2 不支持 SELECT INTO。

```
select * into test from lovelive;
```

创建新表 test，将 lovelive 表所有内容负责到新表 test。

MariaDB、MySQL、Oracle、PostgreSQL 和 SQLite 使用的语法稍有不同：

```
mysql> create table test as select * from lovelive;
Query OK, 20 rows affected (0.50 sec)
Records: 20  Duplicates: 0  Warnings: 0
```

使用 SELECT INTO 注意：

1) 任何 SELECT 选项和子句都可以使用，包括 WHERE 和 ORDER BY；

2) 可利用联结从多个表插入数据；

3) 不管从多少个表中检索数据，数据都只能插入到一个表中。

使用联结将数据插入新表：

```
mysql> create table test as
    -> select prod_id, prod_name, prod_price, vend_name, vend_city
    -> from vendors
    -> inner join products on vendors.vend_id = products.vend_id
    -> order by prod_price;
Query OK, 9 rows affected (0.65 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> select * from test;
+---------+--------------------+------------+----------------+------------+
| prod_id | prod_name          | prod_price | vend_name      | vend_city  |
+---------+--------------------+------------+----------------+------------+
| BNBG01  | Fish bean bag toy  |       3.49 | Doll House Inc. | Dollsville |
| BNBG02  | Bird bean bag toy  |       3.49 | Doll House Inc. | Dollsville |
| BNBG03  | Rabbit bean bag toy |       3.49 | Doll House Inc. | Dollsville |
| RGAN01  | Raggedy Ann        |       4.99 | Doll House Inc. | Dollsville |
| BR01    | 8 inch teddy bear  |       5.99 | Bears R Us     | Bear Town  |
| BR02    | 12 inch teddy bear |       8.99 | Bears R Us     | Bear Town  |
| RYL01   | King doll          |       9.49 | Fun and Games  | London     |
| RYL02   | Queen doll         |       9.49 | Fun and Games  | London     |
| BR03    | 18 inch teddy bear |      11.99 | Bears R Us     | Bear Town  |
+---------+--------------------+------------+----------------+------------+
```

SELECT INTO 是测试新 SQL 语句前进行表复制的很好工具。先复制，在复制的数据上测试 SQL 代码，不会影响实际的数据。

第 16 课　更新和删除数据

✧ 16.1　更新数据

UPDATE 语句更新表中的数据。
基本 UPDATE 语句有 3 部分组成：
1) 要更新的表；
2) 列名和新值；
3) 确定要更新哪些行的过滤条件。

```
mysql> update test
    -> set name='hoshizora rin'
    -> where id=5;
Query OK, 1 row affected (0.13 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

SET 子句用于设置新值。UPDATE 通常需要 WHERE 子句，告诉 DBMS 更新哪些行；否则会将表该列所有行全部修改。

修改多个列：

```
mysql> update test
    -> set name='星空凛',
    ->     bust=80
    -> where id=5;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

删除某列的值，可以设置其为 NULL(该列允许空值)

✧ 16.2　删除数据

DELETE 语句删除表中的数据。用法与 UPDATE 类似。

删除一行：

```
mysql> delete from test
    -> where id = 20;
Query OK, 1 row affected (0.10 sec)
```

友好的外键：
1) 存在外键时，DBMS 使用它们实施引用完整性。如向 products 表中插入一个新产品，DBMS 不允许通过未知的供应商 id 插入，因为 vend_id 列是作为外键连接到 vendors 表的。
2) DBMS 通常可以防止删除某个关系需要用到的行。如要从 products 表中删除一个产品，而这个产品用在 OrderItems 的已有订单中，那么 DELETE 语句将抛出错误并中止。

注意：
1) DELETE 不需要列名或通配符，因为 DELETE 删除的是整行而不是删除列。要删除指定的列，使用 UPDATE 语句；
2) DELETE 语句从表中删除行，但是 DELETE 不删除表本身。

3) 如果想从表中删除所有行，不要使用 DELETE。可使用 TRUNCATE TABLE 语句，它完成相同的工作，而速度更快(因为不记录数据的变动)

```
mysql> TRUNCATE TABLE test;
Query OK, 0 rows affected (0.38 sec) -- 这就是没有记录数据的变动?

mysql> select *from test;
Empty set (0.00 sec)
```

✧ 16.3 更新和删除的指导原则

1) 除非想要更新或删除所有行，否则一定要带 where 子句；
2) 保证每个表都要有主键，尽可能像 where 子句那样使用它；
3) 在 update 和 delete 语句使用 where 子句前，应该先用 select 进行测试，保证过滤条件是正确的；
4) 使用强制实施引用完整性的数据库，这样 DBMS 将不允许删除其数据与其他表相关联的行；
5) 若是 SQL 没有撤销(undo)按钮，应该非常小心地使用 UPDATE 和 DELETE，因为有可能更新了错误的数据或删除了想要的数据。

第 17 课 创建和操纵表

17.1 创建表

创建表的方法：

1) 使用 SQL 的 CREATE TABLE 语句
2) 多数 DBMS 提供交互式创建和管理数据库表的工具，其实际上也是使用 SQL 语句。界面工具自动生成并执行 SQL 语句。

① 表创建基础

create table 创建表，必须给出：

1) 新表名字，在关键字 create table 后给出；
2) 表列的名字和定义，用逗号隔开；
3) 有的 DBMS 还要指定表的位置。

如创建产品表 products：

```
CREATE TABLE products(
    prod_id       CHAR(10)          NOT NULL,
    vend_id       CHAR(10)          NOT NULL,
    prod_name     CHAR(254)         NOT NULL,
    prod_price    DECIMAL(8,2)      NOT NULL,
    prod_desc     VARCHAR(1000)     NULL
);
```

不同 DBMS 的 CREATE TABLE 语法略有不同，比如这条语句在 Oracle、PostgreSQL、SQL Server 和 SQLite 中有效；而对于 MySQL，varchar()必须替换为 text；对于 DB2，必须从最后一列中去掉 NULL。所以不同的 DBMS，要编写不同的表创建脚本。

MySQL 创建 lovelive 表：

```
DROP TABLE IF EXISTS `lovelive`;
CREATE TABLE `lovelive` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(64) NOT NULL,
  `birthday` varchar(10),
  `grade` varchar(1) NOT NULL,
```

```
  `cv` varchar(64) NOT NULL,
  `groupname` varchar(64) NOT NULL,
  `bust` int,
  `waist` int,
  `hip` int,
  `height` int,
  `bloodType` varchar(2),
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=19 DEFAULT CHARSET=utf8;
```

注意：创建新表时，要求指定的表名不存在，如果存在报错。
所以添加创建之前一句：DROP TABLE IF EXISTS `tablename`;
表示如果指定表名存在，则删除后再重新创建。

② 使用 NULL 值
创建表时，指定列为 NULL 表示该列允许 NULL 值；指定列为 NOT NULL 表示该列不能为 NULL，执行 INSERT 语句时该列必须有值。

注意：
1) 不指定 NOT NULL 时，大部分 DBMS 默认指定 NULL。但 DB2 要求显式指定 NULL，不指定会报错；
2) 主键是唯一标识表中每一行的列，必须 NOT NULL；

③ 指定默认值
在插入行时如果不给出值，DBMS 将使用默认值；默认值在 CREATE TABLE 语句中用关键字 DEFAULT 定义。

```
CREATE TABLE 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
);
```

quantity 为订单中每个物品的数量，默认指定 1。

默认值经常用于日期或时间戳列，一般指定为系统日期的函数或变量。

DBMS	**函数/变量**
Access	NOW()
DB2	CURRENT_DATE
MySQL	CURRENT_DATE()
Oracle	SYSDATE
PostgreSQL	CURRENT_DATE
SQL Server	GETDATE()
SQLite	date('now')

✧ 17.2 更新表
使用 ALTER TABLE 语句更新表，不同 DBMS 允许更新的内容差别很大。
需要考虑的事：
1) 理想情况下，不要中表包含数据时对其进行更新。应该在表的设计过程中充分考虑未来可能的需求，避免对表结构做大改动；

2) 所有的 DBMS 都允许给现有的表增加列，不过对所增加列的数据类型(以及 NULL 和 DEFAULT 的使用)有所限制；
3) 许多 DBMS 不允许删除或更改表中的列；
4) 多数 DBMS 允许重命名表中的列；
5) 许多 DBMS 限制对已经填有数据的列进行更改，对未填有数据的列几乎没有限制。

- alter table Vendors add vend_phone CHAR(20);

```
复杂的表结构更改一般需要手动删除过程，它涉及以下步骤:
1. 用新的列布局创建一个新表;
2. 使用 INSERT SELECT 语句从旧表复制数据到新表。有必要的话，可以使用转换函数和计算字段;
3. 检验包含所需数据的新表;
4. 重命名旧表(如果确定，可以删除它);
5. 用旧表原来的名字重命名新表;
6. 根据需要，重新创建触发器、存储过程、索引和外键
```

17.3 删除表
- drop table Customers;

注意：删除表不可撤销。请慎重

17.4 重命名表
每个 DBMS 对表重命名的支持有所不同，具体要参考具体的 DBMS 文档.