SQL基础教程学习笔记

0.文档说明

文档是基于SQL基础教程第2版整理的笔记，该书非常适合新手入门SQL语言；虽然整理了书中大部分核心内容，但需要完整的代码示例和学习还请阅读原书；为了方便整理，大部分SQL代码都写成了一行，实际应用时根据个人代码习惯调整；文档中“- *斜体字* ”均为备注说明；另外，附录中的内容是本人添加的，并非书中所有😄

> 关于笔记的疑问或交流学习可加本人微信`cnicedon`

1.数据库和SQL

DBMS的种类

- HDB (Hierarchical Database) ： 层次数据库
- RDB (Relational Database) ： 关系数据库
- OODB (Object Oriented Database) ： 面向对象数据库
- XMLDB (XML Database) ： XML数据库
- KVS (Key-Value Store) ： 键值存储系统

表的结构

- 表的列称为字段
- 表的行称为记录

SQL概要

- 标准SQL

  本书介绍的均为标准SQL的写法，特定DBMS的SQL语句会单独说明

- SQL语句及种类
  - DDL：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
  - DML：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：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
- SQL书写规则
  - SQL语句以分号`;`结尾
  - SQL语句不区分大小写
  - 常数的书写方式
    - 字符串和日期常数需要使用单引号`'`括起来
    - 数字常数无需单引号，直接写数字即可
  - 单词之间需要使用半角空格或者换行符进行分隔

表的创建(CREATE)

- 创建数据库

  `CREATE DATABASE 数据库名;`

- 创建表

  `CREATE TABLE 表名;`

```
CREATE TABLE 表名
(列名1 数据类型 列的约束,
 列名2 数据类型 列的约束,
 列名3 数据类型 DEFAULT 值,
 。。。
 表的约束1,表的约束2,。。。);
 -------------------------------
 PRIMARY KEY (列名)  ---> 设置主键，可以多个
```

- *表的名称必须以半角英文字母作为开头；表的名称不能重复*
- *其中 DEFAULT为当前列设置默认值*

- 数据类型
  - `INTEGER` 存储整数的列的数据类型，不能存储小数
  - `CHAR` 存储字符串的列的数据类型(字符型)，如“CHAR(10)”，指定最大长度，定长字符串
  - `VARCHAR` 存储字符串的列的数据类型(字符串型)，如“VARCHAR(8)”，可变长字符串
  - `DATE` 存储日期（年月日）的列的数据类型（日期型）

表的删除和更新(DROP/ALTER)

- 删除表

  `DROP TABLE 表名;`

- 添加列

  `ALTER TABLE 表名 ADD COLUMN 列名 约束;`

- 删除列

  `ALTER TABLE 表名 DROP COLUMN 列名;`

  - *常规情况下表和列的删除无法恢复，请在操作前确认*

- 表中插入数据

```
-- 以下代码适用于SQL Server和PostgreSQL
BEGIN TRANSACTION;

INSERT INTO 表名 VALUES ('001','测试','人生','4000','2019-09-15',NULL);
INSERT INTO 表名 VALUES ('002','运营','人类','5000','2019-08-20','400');

COMMIT;
-- 注1：MySQL中第一行为 START TRANSACTION
-- 注2：Oracle和DB2中无需第一行
```

- 变更表名
  - Oracle & PostgreSQL

    `ALTER TABLE 旧表名 RENAME TO 新表名;`

  - SQL Server

    `sp_rename '旧表名','新表名';`

  - MySQL

    `RENAME TABLE 旧表名 to 新表名;`

2.查询基础

SELECT基础

- 查询列

  `SELECT 列名1,列名2 FROM 表名;`

- 查询表中所有列

  `SELECT * FROM 表名;`

- 为列设置别名

  `SELECT 列名1 AS 别名1,列名2 AS 别名2 FROM 表名;`

  `SELECT 列名1 AS “中文列名1”,列名2 AS “中文列名2” FROM 表名;`

  - 设置为中文列名需要加英文双引号

- 常数查询

  `SELECT '名称' AS string, 20 AS number, '2019-09-15' AS date,列名1,列名2 FROM 表名;`

  - SQL语句中若使用字符串或日期常数时，必须使用单引号

- 从结果中删除重复行

  `SELECT DISTINCT 列名 FROM 表名;`

  `SELECT DISTINCT 列名1,列名2 FROM 表名;`

  - DISTINCT关键字只能用在第一个列名之前

  - 使用 DISTINCT时，NULL也被视为一类数据

- 使用WHERE选择记录

  `SELECT 列名1,列名2 FROM 表名 WHERE 条件表达式;`

  - 先通过WHERE查询符合条件记录，再选取出SELECT语句指定的列;

  - WHERE语句须在FROM子句之后

- 注释
  - 单行注释：在 `--` 之后;
  - 多行注释：在 `/*` 和 `*/` 之间

算术运算符和比较运算符

- SQL中的算数运算符

含义	运算符
加法运算	+
减法运算	-
乘法运算	*
除法运算	/

- 算数运算符示例

  `SELECT 列名1,列名2,列名2 * 3 FROM 表名;`

  - 其中列名2的数据类型为整数型时，直接进行数学运算

  - 所有包含 NULL的计算，结果都为 NULL，即使是NULL / 0这样的计算

- SQL中的比较运算符

运算符	含义
=	和 XXX 相等
<>	和 XXX 不相等
>=	大于等于 XXX
>	大于 XXX
<=	小于等于 XXX
<	小于 XXX

- 比较运算符示例

  `SELECT 列名1,列名2 FROM 表名 WHERE 列名 < 值;`

  `SELECT 列名1,列名2 FROM 表名 WHERE 列名 >= '日期';`

  `SELECT 列名1,列名2 FROM 表名 WHERE 列名1 - 列名3 > 值;`

  - 小于某个日期就是在该日期之前；在某日期（含该日期）之后的查询条件时，使用 >=

- 对字符串使用比较运算时
  - 字符串类型的数据原则上按照字典顺序进行排序
  - 该规则适用于长字符串和可变长字符串
- 不能对NULL使用比较运算符

  `SELECT 列名1,列名2 FROM 表名 WHERE 列名2 IS NULL;`

  `SELECT 列名1,列名2 FROM 表名 WHERE 列名2 IS NOT NULL;`

  - 要选取 NULL 记录时用“IS NULL”运算符，选取不是 NULL 记录使用 "IS NOT NULL"运算符

逻辑运算符

- NOT运算符

  `SELECT 列名1,列名2 FROM 表名 WHERE NOT 列名1 > 值;`

  - NOT运算符用来否定某个条件，不能滥用

- AND和OR运算符

  `SELECT 列名1,列名2 FROM 表名 WHERE 列名 = 值 AND 列名3 <= 值;`

  `SELECT 列名1,列名2 FROM 表名 WHERE 列名 = 值 OR 列名3 >= 值;`

  - 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

  - OR运算符：在其两侧的查询条件有一个成立（或都成立）时整个查询条件都成立

  - AND运算符优先于OR运算符；若需要优先执行OR运算符，可使用括号

3.聚合与排序

对表聚合查询

- 聚合函数
  - `COUNT` 记录表中的记录数（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大值
- 计算表中的行数

  `SELECT COUNT(*) FROM 表名;`

  - 计算全部数据的行数

  `SELECT COUNT(列名) FROM 表名;`

  - 计算NULL之外的行数

  - 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；其中星号是COUNT函数所特有的
- 计算合计值

  `SELECT SUM(列名) FROM 表名;`

  `SELECT SUM(列名1),SUM(列名2) FROM 表名;`

  - 计算结果中，聚合函数会将NULL排除再外，但COUNT(*)列外，不会排除NULL
- 计算平均值

  `SELECT AVG(列名) FROM 表名;`

  SELECT AVG(列名1),AVG(列名2) FROM 表名;```

  - 计算某一列或多列的平均值

  - 与SUM函数相同，AVG函数事先会将NULL值排除在外进行计算
- 计算最大值和最小值

  `SELECT MAX(列名1),MIN(列名2) FROM 表名;`

  - 原则上，MAX和MIN函数适用于所有数据类型的列，SUM和AVG函数只适用于数据类型的列
- 使用聚合函数删除重复值

  `SELECT COUNT(DISTINCT 列名) FROM 表名;`

  - 计算去除重复数据后的数据行数

  `SELECT DISTINCT COUNT(列名) FROM 表名;`

  - 先计算数据行数再删除重复数据的结果

对表分组(GROUP BY)

- GROUP BY子句

  `SELECT 列名1,列名2 FROM 表名 GROUP BY 列名1,列名2;`

  - GROUP BY子句的作用是将表中的数据分为多个组处理

  - GROUP BY子句中指定的列称为聚合键或分组列

  - GROUP BY子句必须在FROM子句之后，若有WHERE子句，则在WHERE子句之后

  - 当聚合键中包含NULL的时候，结果会以“不确定”行（空行）的形式显示
- 使用WHERE子句和GROUP BY子句

  `SELECT 列名1,列名2 FROM 表名 WHERE 表达式 GROUP BY 列名,列名1;`

  - WHERE和 GROUP BY并用时的执行顺序：FROM->WHERE->GROUP BY->SELECT
- 与聚合函数和GROUP BY有关的常见问题
    - 使用GROUP BY子句时， SELECT子句中不能出现聚合键之外的列名

      - 在 SELECT子句中出现聚合键之外的列名仅有MySQL支持，而其它DBMS均不支持这种语法
    - 在GROUP BY子句中不能使用 SELECT 子句中定义的别名

- *原因在于SELECT子句在GROUP BY子句之后执行*
- *在 MySQL和 PostgreSQL中使用不会出现错误，但不建议这么做*

- GROUP BY子句执行的结果是随机的，无序的
  - *想要按照某种特定顺序进行排序的话，需要在 SELECT 语句中进行指定*
- 只有SELECT、HAVING和ORDER BY子句中可以使用聚合函数
  - *其它如 WHERE子句中无法使用聚合函数*

为聚合结果指定条件(HAVING)

- HAVING子句

  `SELECT 列名1,列名2 FROM 表名 GROUP BY 列名1,列名2 HAVING 分组结果对应的条件;`

  - *HAVING子句必须在GROUP BY子句之后*
  - *HAVING子句中不能使用SELECT子句中定义的别名*
- HAVING子句的构成要素
  - 常数
  - 聚合函数
  - GROUP BY子句中指定的列名（聚合键）
    - *包含 GROUP BY子句时的 SELECT子句也同样有这三个构成要素*
- WHERE和HAVING中更适合的条件
  - 聚合键所对应的条件应该写在WHERE子句中，而不是HAVING子句中
  - WHERE子句 = 指定行所对应的条件
  - HAVING子句 = 指定组所对应的条件
  - 在WHERE和HAVING子句中都可使用的条件，最好写在WHERE子句中，执行速度更快

对查询结果排序(ORDER BY)

- ORDER BY子句

  `SELECT 列名1,列名2 FROM 表名 ORDER BY 排序基准列1,排序基准列2;`

  - *ORDER BY子句中指定的列名称为排序键*
  - *任何情况下，ORDER BY子句都要写在SELECT子句的尾部*
  - *优先使用左侧的排序键，如果该列存在相同值时，参考右侧排序键*

  `SELECT 列名1,列名2 FROM 表名 ORDER BY 排序基准列1 DESC;`

  - *按照由高到低（降序）进行排列*

  `SELECT 列名1,列名2 FROM 表名 ORDER BY 排序基准列1 ASC;`

  - *按照由低到高（升序）进行排列*
  - *默认（未指定）排序规则时，ORDER BY按照升序排列，即无需写ASC*
- NULL的排序
  - 排序键中包含NULL时，会在开头或末尾进行汇总
- 在排序键中使用别名

  `SELECT 列名1,列名2 AS test,列名3 AS test2 FROM 表名 ORDER BY test,test2;`

  - *SELECT子句执行在GROUP BY之后，ORDER BY之前*
  - *ORDER BY子句中可以使用 SELECT子句中定义的别名*
- ORDER BY子句中可以使用SELECT子句中未包含的列和聚合函数

`SELECT 列名,列名1,列名2 FROM 表名 ORDER BY 列名3;`

`SELECT 列名,列名1 FROM 表名 ORDER BY COUNT(*);`

- 在ORDER BY子句中不要使用列编号

  `SELECT 列名1,列名2,列名3 FROM 表名 ORDER BY 2 DESC,3;`

  - 以上代码是通过SELECT子句中的列编号（从左到右依次为1，2，3...）进行排序，不建议使用

4.数据更新

数据的插入(INSERT)

- INSERT基本语法

  `INSERT INTO 表名 (列名1,列名2,列名3,...) VALUES (值1,值2,值3,...);`

  - 列清单和值清单必须保持一致

  - 原则上，执行一次 INSERT语句会插入一行数据

- 多行INSERT
  - 通常使用的INSERT

    `INSERT INTO 表名 VALUES (值1,值2,值3,...);`

    - 对表进行全列 INSERT时，可省略表名后的列清单，但值的数量要和列的数量保持一致

  - 多行INSERT

    `INSERT INTO 表名 VALUES (值1,值2,值3,...),(值10,值20,值30,...),(值11,值21,值31,...);`

    - 该语法适用于 DB2、SQL、SQL Server、PostgreSQL 和 MySQL，但不适用于Oracle

    ```
    -- Oracle中的多行INSERT
    INSERT ALL INTO 表名 VALUES (值1,值2,值3)
    •          INTO 表名 VALUES (值1,值2,值3)
    •          INTO 表名 VALUES (值1,值2,值3);
    ```

- 插入NULL

  `INSERT INTO 表名 (列名1,列名2,列名3,...) VALUES (值1,NULL,值3,...);`

  - 插入NULL的列不能设置 NOT NULL约束

  - 插入设置为NOT NULL约束的列时会出错，但不影响之前已经插入的数据

  - 不仅是INSERT，DELETE和 UPDATE等语句执行失败同样对表的数据无影响

- 插入默认值
  - 显式方法

    `INSERT INTO 表名 (列名1,列名2,列名3,...) VALUES (值1,DEFAULT,值3,...);`

    - 如果在创建表时同时设定了默认值，就可以在 INSERT 语句中自动为列赋值

  - 隐式方法
    - 插入默认值可不用DEFAULT关键字，只要在列清单和VALUES中省略设定了默认值的列即可
    - 如果省略了没有设定默认值的列，该列的值就会被设定为 NULL
    - 如果省略的是设置了 NOT NULL约束的列， INSERT语句就会出错
- 从其它表中复制数据

`INSERT INTO 表名B (列名1,列名2,列名3) SELECT 列名1,列名2,列名3 FROM 表名A;`

- 将表A中的数据复制到表B中；可在备份时使用

`INSERT INTO 表名B (列名1,列名2,列名3) SELECT 列名1,列名2,列名3 FROM 表名A GROUP BY 列名1;`

- INSERT语句的 SELECT语句中，可以使用 WHERE或 GROUP BY等子句，但使用ORDER BY无效果

数据的删除(DELETE)

- DROP TABLE语句和DELETE语句
  - `DROP TABLE` 完全删除整张表
  - `DELETE` 删除数据（行），但是保留表
- DELETE基本语法

`DELETE FROM 表名;`

- 删除表中的全部数据（行），但保留数据表

- DELETE语句的删除对象并不是表或列，而是记录（行）

`DELETE FROM 表名 WHERE 条件;` 例：`DELETE FROM 表名 WHERE 列名 > 值;`

- 通过 WHERE子句指定对象条件删除部分数据

```
TRUNCATE 表名;
-- 标准SQL中删除表里的数据只有DELETE语句
-- TRUNCATE语句适用于Oracle/SQL Server/MySQL/PostgreSQL
-- TRUNCATE语句只能删除表中的全部数据，不能通过WHERE子句指定条件
```

数据的更新(UPDATE)

- UPDATE基本语法

`UPDATE 表名 SET 列名 = 表达式;`

- UPDATE语句会把列中为NULL的值也更新

`UPDATE 表名 SET 列名 = 表达式 WHERE 条件;`

- 为UPDATE语句指定条件

`UPDATE 表名 SET 列名 = NULL WHERE 条件;`

- UPDATE语句可以将列更新为NULL，俗称NULL清空 （但只限于未设置 NOT NULL约束的列）

- 多列更新

`UPDATE 表名 SET 列名1 = 表达式1,列名2 = 表达式2 WHERE 条件;`

- SET子句支持同时将多个列作为更新对象；适用于所有DBMS

`UPDATE 表名 SET (列名1,列名2) = (表达式1,表达式2) WHERE 条件;`

- 该语句仅适用于PostgreSQL和 DB2

事务

- 事务的概念

  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；就是需要在同一个处理单元中执行一系列更新操作的情况，需要使用事务进行处理。

- 创建事务

```
-- 语法
```

```
事务开始语句;
    DML语句1;
    DML语句2;
    DML语句3;
事务结束语句（COMMIT或ROLLBACK）;

-- 示例
BEGIN TRANSACTION;
UPDATE 表名 SET 列名 = 表达式 WHERE 条件;
UPDATE 表名 SET 列名 = 表达式 WHERE 条件;
COMMIT;
-- 注1：以上代码适用于SQL Server和PostgreSQL
-- 注2：MySQL中第一行为 START TRANSACTION
-- 注3：Oracle和DB2中无需第一行
```

- *在标准 SQL 中并没有定义事务的开始语句，而是由各DBMS自己来定义*

    - COMMIT（提交处理）

      COMMIT 是提交事务包含的全部更新处理的结束指令；一旦提交，就无法恢复到事务开始前的状态，使用前需确认操作

    - ROLLBACK（取消处理）

      ROLLBACK是取消事务包含的全部更新处理的结束指令；一旦回滚，就会恢复到事务开始之前的状态

- ACID特性
    - 原子性(Atomicity):指在事务结束时，其中所包含的更新处理要么全部执行，要么不执行
    - 一致性(Consistency):指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NOT NULL约束
    - 隔离性(Isolation):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
    - 持久性(Durability):指在事务（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；为了保证持久性，可将执行记录（日志）保存

5.复杂查询

视图

- 视图和表
    - 表中存储的是实际数据，而视图中保存的是从表中取出数据所使用的SELECT语句
    - 应该将经常用的SELECT语句做成视图，方便调用;视图的本质是SELECT语句
- 创建视图
    - 视图语法

      `CREATE VIEW 视图名称（视图列名1,视图列名2,...) AS SELECT语句`

      - SELECT语句中的列的顺序需和视图中列的顺序相同

```sql
-- 视图示例
CREATE VIEW 视图名A (视图列1,视图列2)
AS
SELECT 列1,COUNT(*) FROM 表名 GROUP BY 列1;

-- 使用视图
SELECT 列1,列2 FROM 视图名A;
```

- 多重视图

```sql
-- 视图嵌套示例
CREATE VIEW 视图名A (视图列1,视图列2)
AS
SELECT 列1,列2 FROM 视图名B WHERE 条件;
```

- 视图的限制1

  通常情况下，定义视图时不能使用ORDER BY语句；个别DBMS如PostgreSQL中可以使用

- 视图的限制2

  如果定义视图的SELECT语句能够满足某些条件，那么这个视图就可以被更新：

  - SELECT子句中未使用DISTINCT
  - FROM子句中只有一张表
  - 未使用GROUP BY子句或未使用HAVING子句

- 删除视图

  `DROP VIEW 视图名 (视图列名1,视图列名2,...);`

  `DROP VIEW 视图名 CASCADE;`

  - 在PostgreSQL中如果删除多重视图，需要用 CASCADE选项来删除关联视图，否则删除时会出错

子查询

- 基本概念

  子查询就是将用来定义视图的SELECT语句直接用于FROM子句当中

  `SELECT 列名1,列名2 FROM (子查询SELECT语句) AS 子查询名称;`

  - 子查询作为内层查询会优先执行

  `SELECT 列名1,列名2 FROM (SELECT * FROM (子查询SELECT语句) AS 子查询名称1 WHERE 条件) AS 子查询名称2;`

  - 原则上子查询的层数没有限制，可以无限嵌套，但尽量避免使用多层嵌套

- 标量子查询

  标量子查询就是返回单一值的子查询；必须而且只能返回1行1列的结果

  `SELECT 列名1,列名2 FROM 表名 WHERE 列名 > (SELECT AVG(列名) FROM 表名);`

  - 标量子查询的返回值可用在 = 或 <> 这样需要单一值的比较运算符中

- 标量子查询的书写位置和注意事项
  - 并不局限WHERE子句，通常任何可以使用单一值的位置都可使用；即，能够用常数或列名的地方，无论是 SELECT、GROUP BY、HAVING，还是ORDER BY子句，几乎所有的地方都可以使用
  - 标量子查询不能返回多行结果，否则就是一个普通的子查询

关联子查询

- 普通子查询和关联子查询区别

  在细分的组内进行比较时，需要使用关联子查询

  ```
  SELECT 列名1,列名2,列名3 FROM 表名 AS P1 WHERE 列名3 > (SELECT AVG(列名3) FROM 表名 AS P2 WHERE P1.列名1 = P2.列名1 GROUP BY 列名1);
  ```

  - 上面代码中的GROUP BY子句不是必须的；P1和P2分别为表的别名

  - 结合条件（如 WHERE）一定要写在子查询内，因为关联别名（如P1和P2）是有作用范围的

  > 关联子查询是以一种循环检查的方式来查询的，也就是外部查询先取一条记录，然后该记录传进内部查询，用内部查询来筛选此条记录，如果符合内部查询的条件，再返回经内部查询执行过的该记录到外部查询中，取相应的字段显示。

6.函数、谓词、CASE表达式

各种各样的函数

- 函数的种类
  - 算术函数（用来进行数值计算的函数）
  - 字符串函数（用来进行字符串操作的函数）
  - 日期函数（用来进行日期操作的函数）
  - 转换函数（用来转换数据类型和值的函数）
  - 聚合函数（用来进行数据聚合的函数）
- 算数函数

  除了基本的“ `+` 加法 `-` 减法 `*` 乘法 `/` 除法 ”外的算数函数

  - ABS（绝对值）

    ```
    ABS(数值)
    ```

    ```
    SELECT a,ABS(a) AS abs_col FROM 表名;
    ```

    - 列abs_1显示的就是列a通过ABS函数计算的绝对值结果

    - ABS函数中参数为NULL时，结果也是NULL

  - MOD（求余）

    ```
    MOD(被除数,除数)
    ```

    ```
    SELECT a,b,MOD(a,b) AS mod_col FROM 表名;
    ```

    - 只能对整数类型的列使用 MOD函数，适用于大部分DBMS

    ```
    SELECT a,b,a % b AS mod_col FROM 表名;
    ```

    - SQL Server中不支持MOD函数，而是采用运算符"%"来计算余数

  - ROUND（四舍五入）

    ```
    ROUND(对象值,保留小数的位数)
    ```

    ```
    SELECT m ROUND(m,2) AS round_col FROM 表名;
    ```

    - 对m列的数值进行2位数的四舍五入处理

- 字符串函数
  - ||（拼接）

    ```
    字符串1 || 字符串2
    ```

    ```
    SELECT 列1,列2,列1 || 列2 AS str FROM 表名;
    ```

    - 拼接列1和列2的字符串并显示在str列中；该函数可以拼接多个字符串，不仅是两个

- *包含NULL时，得到的结果也是NULL；另外该函数并不适用于所有DBMS*

```
-- SQL Server中使用“+”运算符;另外,SQL Server2012及其以后版本中也支持CONCAT函数
SELECT 列1,列2,列3 列1 + 列2 + 列3 AS str_con FROM 表名;

-- MySQL中使用CONCAT函数
SELECT 列1,列2,列3,CONCAT(列1,列2,列3) AS str_con FROM 表名;
```

- LENGTH（字符串长度）

  `LENGTH(字符串)`

  `SELECT 列1,LENGTH(列1) AS len_str FROM 表名;`

  - *计算列1中的字符串长度并显示在len_str列中，该函数对SQL Server不适用*
  - *MySQL中存在自有函数CHAR_LENGTH来计算字符串长度*

  `SELECT 列1,LEN(列1) AS len_str FROM 表名;`

  - *SQL Server中使用LEN函数来计算字符串长度*

- LOWER（小写转换）

  `LOWER(字符串)`

  `SELECT 列1,LOWER(列1) AS low_str FROM 表名;`

  - *将列1中的字符串转换为小写并显示在low_str列中*
  - *LOWER函数只能应用在英文字母中，英文字母以外不适用；同时不影响原来就是小写的字符*
  - *UPPER函数为大写转换，规则同LOWER函数*

- REPLACE（字符串替换）

  `REPLACE(对象字符串,替换前的字符串,替换后的字符串)`

  `SELECT 列1,列2,列3,REPLACE(列1,列2,列3) AS rep_str FROM 表名;`

  - *把列1中包含列2的字符串替换成列3中的字符串，并显示在rep_str列中*
  - *REPLACE函数可以将字符串的一部分替换为其他的字符串*

- SUBSTRING（字符串的截取）

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

  `SELECT 列1,SUBSTRING(列1 FROM 3 FOR 2) AS sub_str FROM 表名;`

  - *截取列1的字符串中第三位和第四位字符，并显示在sub_str列中*
  - *SUBSTRING函数可以截取出字符串中的一部分字符串,截取的起始位置从字符串最左侧开始计算*
  - *以上代码仅适用于PostgreSQL和 MySQL 中*

```
-- SQL Server中截取的语法
SUBSTRING(对象字符串,截取的起始位置,截取的字符数)
-- Oracle和DB2中截取的语法
SUBSTR(对象字符串,截取的起始位置,截取的字符数)
```

- 日期函数
  - CURRENT_DATE（当前日期）

    `SELECT CURRENT_DATE;`

- *返回 SQL的执行日期；仅适用于PostgreSQL和 MySQL中*

```sql
-- SQL Server使用CURRENT_TIMESTAMP函数来获得当前日期
SELECT CAST(CURRENT_TIMESTAMP AS DATE) AS CUR_DATE;
-- 使用CAST函数将CURRENT_TIMESTAMP转换为日期类型

-- Oracle中使用该函数时，需要在FROM子句中指定临时表（DUAL）
SELECT CURRENT_DATE FROM dual;

-- DB2中需要在CRUUENT和DATE之间添加半角空格，并指定临时表SYSIBM.SYSDUMMY1
SELECT CURRENT DATE FROM SYSIBM.SYSDUMMY1;
```

- CURRENT_TIME（当前时间）

`SELECT CURRENT_TIME;`

- *返回SQL的执行时间；仅适用于PostgreSQL和 MySQL中*

```sql
-- SQL Server使用CURRENT_TIMESTAMP函数来获得当前日期
SELECT CAST(CURRENT_TIMESTAMP AS TIME) AS CUR_TIME;
-- 使用CAST函数将CURRENT_TIMESTAMP转换为时间类型

-- Oracle中需要在FROM子句中指定临时表（DUAL），得到的结果还包含日期
SELECT CURRENT_TIMESTAMP FROM dual;

-- DB2中需要在CRUUENT和TIME之间添加半角空格，并指定临时表SYSIBM.SYSDUMMY1
SELECT CURRENT TIME FROM SYSIBM.SYSDUMMY1;
```

- CURRENT_TIMESTAMP（当前日期和时间）

`SELECT CURRENT_TIMESTAMP;`

- *获取当前的日期和时间；适用在SQL Server，PostgreSQL和 MySQL中*

```sql
-- Oracle中需要在FROM子句中指定临时表（DUAL）
SELECT CURRENT_TIMESTAMP FROM dual;

-- DB2中需要在CRUUENT和TIME之间添加半角空格，并指定临时表SYSIBM.SYSDUMMY1
SELECT CURRENT TIMESTAMP FROM SYSIBM.SYSDUMMY1;
```

- EXTRACT（截取日期元素）

`EXTRACT(日期元素 FROM 日期)`

`SELECT CURRENT_TIMSTAMP,EXTRACT(YEAR FROM CURRENT_TIMESTAMP) AS year,EXTRACT(MONTH FROM CURRENT_TIMESTAMP) AS month,EXTRACT(DAY FROM CURRENT_TIMESTAMP) AS day,EXTRACT(HOUR FROM CURRENT_TIMSTAMP) AS hour;`

- *该函数返回的不是日期类型，而是数值类型；以上代码仅适用于PostgreSQL和 MySQL中*

```sql
-- SQL Server中使用DATEPART函数
SELECT CURRENT_TIMSTAMP,DATEPART(YEAR FROM CURRENT_TIMESTAMP) AS year,DATEPART(MONTH FROM CURRENT_TIMESTAMP) AS month,DATEPART(DAY FROM CURRENT_TIMESTAMP) AS day,DATEPART(HOUR FROM CURRENT_TIMSTAMP) AS hour;

-- Oracle中使用需要指定临时表（dual）
-- DB2要在CRUUENT和TIMESTAMP间加半角空格，并指定临时表SYSIBM.SYSDUMMY1
```

- 转换函数
  - CAST（类型转换）

    `CAST(转换前的值 AS 想要转换的数据类型)`

```sql
-- 以下代码是将字符串类型转为数值类型
-- SQL Server和PostgreSQL代码
SELECT CAST('0001' AS INTEGER) AS int_col;
-- MySQL代码
SELECT CAST('0001' AS SIGNED INTEGER) AS int_col;
-- Oracle代码
SELECT CAST('0001' AS INTEGER) AS int_col FROM DUAL;
-- DB2代码
SELECT CAST('0001' AS INTEGER) AS int_col FROM SYSIBM.SYSDUMMY1;
```

```sql
-- 以下代码是将字符串类型转为日期类型
-- SQL Server、PostgreSQL和MySQL代码
SELECT CAST('2019-09-21' AS DATE) AS date_col;
-- Oracle代码
SELECT CAST('2019-09-21' AS DATE) AS date_col FROM DUAL;
-- DB2代码
SELECT CAST('2019-09-21' AS DATE) AS date_col FROM SYSIBM.SYSDUMMY1;
```

  - COALESCE（将NULL转为其它值）

    `COALESCE(数据1,数据2,数据3,...)`

    - COALESCE函数会返回可变参数中左侧开始第1个不是NULL的值；参数个数可变，可无限增加

```sql
-- SQL Server、PostgreSQL和MySQL代码
SELECT COALESCE(NULL,1) AS col_1,COALESCE(NULL,'test',NULL) AS
col_2,COALESCE(NULL,NULL,'2019-09-28') AS col_3;
-- Oracle代码
SELECT COALESCE(NULL,1) AS col_1,COALESCE(NULL,'test',NULL) AS
col_2,COALESCE(NULL,NULL,'2019-09-28') AS col_3 FROM DUAL;
-- DB2代码
SELECT COALESCE(NULL,1) AS col_1,COALESCE(NULL,'test',NULL) AS
col_2,COALESCE(NULL,NULL,'2019-09-28') AS col_3 FROM
SYSIBM.SYSDUMMY1;
```

谓词

- 什么是谓词

  谓词可以称为函数的一种，是需要满足特定条件的函数，该条件就是返回值是真值

- LIKE谓词（字符串的部分一致查询）
  - 前方一致：选取出作为查询条件的字符串与查询对象字符串起始部分相同的记录的查询方法

    `SELECT * FROM 表名 WHERE 列1 LIKE 'aaa%';`

    - 取出列名1中前方包含aaa的字符串的记录

    - 语句中 %代表0字符以上的任意字符串
  - 中间一致：选取出查询对象字符串中含有作为查询条件的字符串的记录的查询方法

    `SELECT * FROM 表名 WHERE 列1 LIKE '%aaa%';`

- *取出列1中包含aaa字符的记录*

  - 后方一致：选取出作为查询条件的字符串与查询对象字符串的末尾部分相同的记录的查询方法

    ```
    SELECT * FROM 表名 WHERE 列1 LIKE '%aaa';
    ```

    - *取出列1中以aaa字符串结尾的记录*

  - 使用"_"(下划线)，代表任意1个字符

    ```
    SELECT * FROM 表名 WHERE 列1 LIKE 'aaa__';
    ```

    - *取出列1中包含aaa字符同时加任意2个字符的记录*

- BETWEEN谓词（范围查询）

  ```
  SELECT 列1,列2 FROM 表名 WHERE 列2 BETWEEN 值1 AND 值2;
  ```

  - *选取介于两个值之间的数据范围内的值；执行结果中是否包含值1和值2的临界值取决于DBMS*
  - *值可以是数值、文本或日期*

  ```
  SELECT 列1,列2 FROM 表名 WHERE 列2 NOT BETWEEN 值1 AND 值2;
  ```

  - *选取介于两个值范围之外的值*

- IS NULL/IS NOT NULL（判断是否为NULL）

  ```
  SELECT 列1,列2 FROM 表名 WHERE 列2 IS NULL;
  ```

  - *取出列2中为NULL的记录*

  ```
  SELECT 列1,列2 FROM 表名 WHERE 列2 IS NOT NULL;
  ```

  - *取出列2中不是NULL的记录*

- IN谓词（OR的简便用法）

  ```
  SELECT 列1,列2 FROM 表名 WHERE 列2 IN(值1,值2,值3);
  ```

  - *取出列2中值为值1，值2，值3的记录*

  ```
  SELECT 列1,列2 FROM 表名 WHERE 列2 NOT IN(值1,值2,值3);
  ```

  - *取出列2中不是值1，值2，值3的记录*
  - *IN和 NOT IN不能取出NULL数据*

  - 使用子查询作为IN谓词的参数

    ```
    SELECT 列1,列2 FROM 表名 WHERE 列3 IN (SELECT语句)
    ```

    - *括号中的 SELECT语句即子查询；在 IN前面加 NOT即为 NOT IN语句*

- EXIST谓词

  ```
  SELECT 列1,列2 FROM 表名 AS 表别名 WHERE EXIST (关联子查询);
  ```

  - *EXIST是只有1个参数的谓词，通常使用关联子查询作为参数*
  - *作为EXIST参数的子查询中经常会使用 "SELECT *"*
  - *NOT EXIST与EXIST相反，当"不存在"满足子查询中指定条件的记录时返回真*

CASE表达式

- CASE表达式语法

```
-- 搜索CASE表达式
CASE WHEN 求值表达式 THEN 表达式
     WHEN 求值表达式 THEN 表达式
     ...
     ELSE 表达式
END
```

- 从第一个WHEN开始执行，若为真，返回THEN子句中的表达式，执行结束；若为假，执行下一个WHEN，若都为假，则执行ELSE中的表达式，执行结束

```
-- CASE代码示例
SELECT 列1,
    CASE WHEN 列2 = 值 THEN '值' || 列名
         WHEN 列2 = 值 THEN '值' || 列名
         ELSE NULL
    END AS 别名
FROM 表名;
-- ELSE NULL是返回NULL的意思；如果ELSE不写，默认为ELSE NULL
-- CASE表达式中END不能省略
```

- Oracle中提供 DECODE，MySQL中为IF这样的特定CASE表达式的简化函数，具体见DBMS说明

7.集合运算

表的加减法

- 表的加法（UNION）

  `SELECT 列1,列2 FROM 表1 UNION SELECT 列1,列2 FROM 表2;`

  - UNION是并集运算；结果中会去除重复记录

- 集合运算的注意事项
  - 作为运算对象的记录中的列数必须相同；
  - 作为运算对象的记录中列的数据类型必须一致；

    - 如果一定要使用不同数据类型的列时可使用CAST转换函数
  - 可以使用任何形式SELECT语句，但ORDER BY子句只能在最后使用一次

    `SELECT 列1,列2 FROM 表1 WHERE 列3 = 值 UNION SELECT 列1,列2 FROM 表2 WHERE 列3 = 值 ORDER BY 列1;`

- 包含重复行的集合运算

  `SELECT 列1,列2 FROM 表1 UNION ALL SELECT 列1,列2 FROM 表2;`

  - 只要在UNION后加上ALL选项即可保留重复记录

- 选取表中的公共部分（INTERSECT）

  `SELECT 列1,列2 FROM 表1 INTERSECT SELECT 列1,列2 FROM 表2 ORDER BY 列1;`

  - INTERSECT是交集运算；结果中会去除重复记录，若需保留重复行，同样是使用ALL选项

- 记录的减法（EXCEPT）

  `SELECT 列1,列2 FROM 表1 EXCEPT SELECT 列1,列2 FROM 表2 ORDER BY 列1;`

  - EXCEPT是差集运算，执行结果包含表1中的记录去除表2中记录剩余的部分

- *Oracle中使用 MINUS而不是EXCEPT，MySQL中不支持EXCEPT*

联结

- 内联结（INNER JOIN）

  `SELECT 表1.列1,表1.列2,表2.列1,表2.列2 FROM 表1 INNER JOIN 表2 ON 表1.列3 = 表2.列3;`

  - *通过2个表中都包含的列作为联结，将只存在1个表内的列汇集到同一个结果中*

    - 进行联结时FROM子句中需要使用多张表；表名可以使用别名，以增加易读性
    - ON用来指定联结条件；代码中的列3称为联结键；指定多个键时可使用AND或OR；内联结中必须使用ON子句，并且要写在FROM和WHERE之间
    - 使用联结时，SELECT子句中的列需要按照`表名.列名`的方式书写

  `SELECT 表1.列1,表1.列2,表2.列1,表2.列2 FROM 表1 INNER JOIN 表2 ON 表1.列3 = 表2.列3 WHERE 表1.列1 = 值;`

  - *内联结可以结合WHERE、GROUP BY、HAVING和ORDER BY等工具使用*

- 外联结（OUTER JOIN）

  `SELECT 表1.列1,表1.列2,表2.列1,表2.列2 FROM 表1 RIGHT OUTER JOIN 表2 ON 表1.列3 = 表2.列3;`

  - 外联结会选取出单张表中的全部信息
  - 外联结中使用LEFT和RIGHT来指定主表，二者得到的结果相同；使用LEFT时左侧的表为主表

- 3张以上的表联结

```sql
SELECT 表1.列1,表1.列2,表2.列1,表2.列2,表3.列1 FROM 表1
INNER JOIN 表2 ON 表1.列3 = 表2.列3
      INNER JOIN 表3 ON 表1.列3 = 表3.列3
WHERE 表3.列2 = 值;
-- 代码使用内联结，外联结同理；表的数量可以不断增加
```

- 交叉联结（CROSS JOIN）

  `SELECT 表1.列1,表1.列2,表1.列3,表2.列1 FROM 表1 CROSS JOIN 表2;`

  - CROSS JOIN(笛卡尔积)是对两张表中的全部记录进行交叉组合（2张表中的行数乘积），实际应用极少

8.SQL高级处理

窗口函数

- 窗口函数概念

  窗口函数也称为OLAP(OnLine Analytical Processing)函数；OLAP是对数据库数据进行实时分析处理

  > 书中介绍MySQL5.7不支持窗口函数，但从MySQL8.0开始已经支持

- 窗口函数语法

  `窗口函数 OVER ([PARTITION BY 列清单] ORDER BY 排序用列清单)`

  - *其中“[]”里的内容可以省略；PARTITION BY进行分组，ORDER BY进行排序*

    - 能够作为窗口函数使用的函数

    - 聚合函数：`SUM` `AVG` `COUNT` `MAX` `MIN`
    - 专用窗口函数：`RANK` `DENSE_RANK` `ROW_NUMBER` 等
- 使用RANK函数

  `SELECT 列1,列2,列3 RANK () OVER (PARTITION BY 列2 ORDER BY 列3) AS 别名 FROM 表名;`

  - PARTITION BY能够设定排序的对象范围；ORDER BY指定按照哪一列、何种顺序进行排序

  - 窗口函数兼具分组和排序两种功能
- 专用窗口函数种类
    - `RANK函数` 计算排序时，如果存在相同位次的记录，则会跳过之后的位次

      例：有3条记录排在第1位时：1位、1位、1位、4位...
    - `DENSE_RANK函数` 计算排序时，即使存在相同位次的记录，也不会跳过之后的位次

      例：有3条记录排在第1位时：1位、1位、1位、2位...
    - `ROW_NUMBER函数` 赋予唯一的连续位次

      例：有3条记录排在第1位时：1位、2位、3位、4位...

      ```sql
      -- 专用窗口函数代码示例
      SELECT 列1,列2,列3,
          RANK () OVER (ORDER BY 列3) AS 别名1,
          DENSE_RANK () OVER (ORDER BY 列3) AS 别名2,
          ROW_NUMBER () OVER (ORDER BY 列3) AS 别名3
      FROM 表名;
      -- 由于专用窗口函数无需参数，因此通常括号中都是空的
      ```
- 窗口函数的适用范围

  原则上窗口函数只能在SELECT子句中使用
- 作为窗口函数使用的聚合函数

  `SELECT 列1,列2,列3,SUM(列3) OVER (ORDER BY 列1) FROM 表名;`

  - 其它聚合函数用法同理，替换SUM即可
- 计算移动平均

  窗口函数包含在窗口中指定更详细的汇总范围的备选功能，该备选功能中的汇总范围称为框架

  `SELECT 列1,列2,列3,AVG(列3) OVER (ORDER BY 列1 ROWS 2 PRECEDING) AS 别名 FROM 表名;`
    - 指定框架（汇总范围）

      上述代码使用ROWS（行）和PRECEDING（之前）关键字，将框架指定为“截止到之前XX行”，这样的统计方法称为移动平均；用关键字FOLLOWING（之后）替换PRECEDING，就可以指定“截止到之后XX行”
    - 将当前记录的前后行作为汇总对象

      `SELECT 列1,列2,列3,AVG(列3) OVER (ORDER BY 列1 ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS 别名 FROM 表名;`

      - “1 PRECEDING”（之前1行）和“1 FOLLOWING”（之后1行）的区间作为汇总对象
    - 两个ORDER BY

      `SELECT 列1,列2,列3 RANK () OVER (ORDER BY 列3) AS 别名 FROM 表名 ORDER BY 别名;`

      - 在语句末尾使用 ORDER BY子句对结果进行排序

GROUPING运算符

GROUPING运算符包含3种：`ROLLUP` `CUBE` `GROUPING SETS`

- 同时得到合计行

```
-- 计算出合计行和汇总结果再通过UNION ALL连接
SELECT '合计' AS 列1,SUM(列2) FROM 表名1
UNION ALL
SELECT 列1,SUM(列2) FROM 表名1
GROUP BY 列1;
-- 不支持GROUPING时使用，繁琐且性能差
```

- 同时得出合计和小计（ROLLUP）

  `SELECT 列1,SUM(列2) AS 别名 FROM 表名 GROUP BY ROLLUP(列1);`

  `SELECT 列1,列3,SUM(列2) AS 别名 FROM 表名 GROUP BY ROLLUP(列1,列3);`

  - MySQL中GROUP BY子句应该写为“GROUP BY 列1 WITH ROLLUP”

- 使用GROUPING函数来判断NULL

  `SELECT GROUPING(列1),GROUPING(列2),SUM(列3) FROM 表名 GROUP BY ROLLUP(列1,列2);`

  - 执行结果中产生NULL时返回 1，其他情况返回 0

```
SELECT CASE WHEN GROUPING(列1) = 1 THEN '名称1 合计' ELSE 列1 END,
       CASE WHEN GROUPING(列2) = 1 THEN '名称2 合计' ELSE 列2 END,SUM(列3)
FROM 表名 GROUP BY ROLLUP(列1,列2);
-- 插入适当的字符来方便查阅结果的显示
```

- 比ROLLUP多维度（CUBE）

  `SELECT 列1,列3,SUM(列2) AS 别名 FROM 表名 GROUP BY CUBE(列1,列3);`

  - MySQL中GROUP BY子句应该写为“GROUP BY 列1,列3 WITH CUBE”

- 从ROLLUP或CUBE的结果中取出部分记录（GROUPING SETS）

  `SELECT 列1,列3,SUM(列2) AS 别名 FROM 表名 GROUP BY GROUPING SETS(列1,列3);`

9.通过应用程序连接数据库

本章涉及编程语言的环境配置和编写，笔记省略，如需学习请阅读原书

附录1：常用DBMS列表

- DBMS排行榜
- Oracle

  官方网址 下载 文档
- MySQL

  官方网址 下载 文档
- SQL Server

  官方网址 下载 文档

- PostgreSQL

  官方网址 下载 文档
- DB2

  官方网址 下载
- MongoDB

  官方网址 下载 文档

附录2：学习资源

图书

针对DBMS的书籍太多，列部分SQL和数据库相关书籍供参考，点击书名可直达豆瓣图书页

- SQL相关

  SQL必知必会（第四版）

  SQL语言艺术

  SQL权威指南（第四版）

  SQL反模式

  深入浅出SQL
- 数据库相关

  数据库系统概念（第六版）

  数据库系统实现（第二版）

网站

- SQL在线学习网站

  W3SCHOOL的SQL教程

  廖雪峰的SQL教程

  菜鸟教程中的SQL教程

  SQL在线指南
- 程序员爱逛的地方

Github	StackOverFlow	StackExchange	CodeProject
Hacker News	SourceForge	W3C	SegmentFault
InfoQ	CSDN社区	博客园	掘金
ChinaUnix	ITPUB	V2EX	开源中国

- 我的最爱

  Google

  Wikipedia

  Youtube

`文档由Typora撰写并导出`

EOF