第3章 SQL语言

> 计科2201zzy友情分享😎😋

3.1 基本概念

- 结构化查询语言 (Structed Query Language, SQL)是关系数据库的标准语言。
- 读音：/'si:kwəl/, 其实是sequel单词的读音。
- 支持SQL的关系数据库管理系统具有三级模式结构
  - 外模式：若干视图和部分基本表
    - 视图：从一个或几个基本表导出的表，是一个虚表。
  - 模式：若干基本表
  - 内模式：若干存储文件

3.1.1 SQL的特点

- 综合统一：集数据定义语言、数据操纵语言、数据控制语言的功能于一体
- 高度非过程化：只需要提出“做什么”，而无需指明“怎么做”，因此无需了解存取路径等细节。
- 面向集合的操作方式：增删改查的对象都可以是元组的集合而非单条记录
- 以同一种语法结构提供多种使用方式：既可以独立使用，也可以嵌入到高级语言程序中。
- 语言简洁，易学易用，核心功能只用了9个动词

SQL功能	动词
数据查询	`SELECT`
数据定义	`CREATE, DROP, ALTER`
数据操纵	`INSERT, UPDATE, DELETE`
数据控制	`GRANT, REVOKE`

3.2 数据定义

操作对象	创建: `CREATE`	删除: `DELETE`	修改: `ALTER`	
模式: `SCHEMA`	✓	✓		
表: `TABLE`	✓	✓	✓	
视图: `VIEW`	✓	✓		
索引: `INDEX`	✓	✓	✓	

3.2.1 模式的定义与删除

- 一个模式下通常包括多个表、视图和索引等数据库对象。可以理解为一个命名空间。
- 定义模式实际上定义了一个命名空间，在这个空间中可以进一步定义该模式包含的数据库对象，如基本表等。
- 模式定义语句
  - `CREATE SCHEMA <模式名> AUTHORIZATION <用户名>`
  - 未指定模式名，则模式名隐含为用户名
- 模式删除语句
  - `DROP SCHEMA <模式名> <CASCADE | RESTRICT>`

3.2.2 基本表的定义与删除

定义基本表

```
CREATE TABLE <表名>
(
        <列名> <数据类型> [<列级完整性约束条件>]
        [,<列名> <数据类型> [<列级完整性约束条件>]]
        ...
        [,<表级完整性约束条件>]
);
```

常用完整性约束

- 主码约束：`PRIMARY KEY`
-
  - 建立时会默认建立此field的索引
  - 此primary key可以作为另外表的foreign key
  - 同时包含非空约束
- 唯一性约束：`UNIQUE`
  - 可以为空
- 非空值约束：`NOT NULL`

学生-课程数据库中基本表的定义

```
CREATE TABLE Student
(
        Sno CHAR(9) PRIMARY KEY,
        Sname CHAR(20) UNIQUE,
        Ssex CHAR(2),
        Sage SMALLINT,
        Sdept CHAR(20)
);
```

```
CREATE TABLE Course
(
        Cno CHAR(4) PRIMARY KEY,
        Cname CHAR(40) NOT NULL,
        Cpno CHAR(4),    /*先修课*/
        Ccredit SMALLINT,
        FOREIGN KEY (CPno) REFERENCES Course(Cno)
                /*表级完整性约束条件，Cpno是Cno的外码*/
);

CREATE TABLE SC
(
        Sno CHAR(9),
        Cno CHAR(4),
        Grade SMALLINT,
        PRIMARY KEY (Sno,Cno),
                /*主码由两个属性构成，必须作为表级完整性定义*/
        FOREIGN KEY (Sno) REFERENCES Student(Sno),
        FOREIGN KEY (Cno) REFERENCES Course(Cno)
);
```

删除基本表

```
DROP TABLE [RESTRICT|CASCADE];
```

- 选择RESTRICT，则该表的删除是有限制条件的。若存在依赖改表的对象(视图、触发器、存储过程或函数)或该表被其他表的约束所引用(CHECK, FOREIGN KEY等)，则此表不能被删除。
- 若选择CASCADE，则相关依赖对象会随着该基本表一起被删除。
- 缺省情况是RESTRICT

3.2.3 索引的建立与删除

- 索引是加快查询速度的有效手段。
- 根据表的需要进行建立，后期维护为DBMS自动操作。

```
CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
ON <表名> (<列名>[<次序>] [,<列名> [<次序>]] ...);
```

- `<次序>` 指定索引值的排列次序。 `ASC` 表示升序， `DESC` 表示降序。
- UNIQUE表明此索引的每一个索引值只对应唯一的数据记录。
- CLUSTER表示要建立的索引是聚簇索引。

3.3 数据查询 - SELECT语句

> 本章节中语句使用学生-课程数据库作为示例

3.3.1 单表查询

语句格式

```
SELECT [ALL|DISTINCT] <目标列表达式> [, <目标列表达式>]
FROM <表名或视图名> [,<表名或视图名>]
[WHERE <条件表达式>]
[GROUP BY <列名1> [HAVING <条件表达式>]]
[ORDER BY <列名2> [ASC|DESC] [,<列名3> [ASC|DESC] ] ]
```

- `ALL|DISTINCT`：可选项，不取消制定列中重复值or去重，默认为 `ALL`
- `ASC|DESC`：按某一列对返回的数据进行排序，按照升序or降序

选择表中的若干列

- 通过 `SELECT` 后的 `<目标列表达式>`，可以指定要查询的属性列。
  - 若需要查询全部列，则将该部分指定为 `*` 即可。
  - 目标列表达式不仅可以是表中的属性列，也可以表达式。
    - 算术表达式。如 `SELECT 2024-Sage FROM Student;` 可以查询学生的出生年份、
    - 字符串常量。如 `SELECT 'Name:',Sname FROM Student;` 此时该列直接返回该字符串 `Name:`
    - 函数。如 `SELECT Lower(Sdept)`，返回对应属性列字符串转换为小写之后的结果。
- 可以通过指定别名来改变查询结果的列标题。
  - 指定的标题位于目标列表达式之后，用空格隔开。
  - 如 `SELECT 2024-Sage BIRTHDAY FROM Student;` 中的 `BIRTHDAY`

选择表中的若干元组

- 使用 `DISTINCT` 消除取值重复的行。
  - 两个元组在某些列上具有相同值，若只取这些列，则可能会得到相同的行。
  - 在 `SELECT` 后加上 `DISTINCT` 可以消除它们。
  - 如 `SELECT DISDINCT Sno FROM SC;` 可以查询有选课记录的所有学生，同时避免重复。
  - 和 `DISTINCT` 相对应的关键词是 `ALL`，也是默认情况（保留所有取值重复行）。
- 使用 `WHERE` 查询满足条件的元组。常用查询条件见下表。
  - 比较大小
    - 等于某值：`SELECT Sname FROM Student WHERE Sdept='CS';` 查询学生名字，限定学生范围为计算机系

        - 比较大小：`SELECT DINTINCT Sno FROM SC WHERE Grade<60;` 查询成绩不合格的学生学号
    - 确定范围：`SELECT Sname From Student WHERE Sage BETWEEN 20 AND 23;` 查询年龄在20-23岁之间的学生名字
    - 确定集合：`SELECT Sname Ssex FROM Student WHERE Sdept IN ('CS','MA','IS');` 查询这三个专业所有学生的名字
    - 字符匹配：
        - `LIKE` 可以用来进行字符串的匹配，一般语法格式如下
            - `[NOT] LIKE '<匹配串>' [ESCAPE '<换码字符>']`
            - 它表示查找指定属性列值与匹配串相匹配的元组。匹配串可以是一个完整字符串，也可以包含通配符 `%` 和 `_`
                - `%`：代表任意长度(可以为0)的字符串。
                - `_`：代表任意单个字符。
            - 换码字符定义了一个转义字符，跟在这个字符后面的 `%` 或者 `_` 不在受到通配规则影响，而是按照原始字符匹配。
        - 例：
            - `SELECT * FROM Student WHERE Sname LIKE '刘%'` 所有刘姓同学
            - `SELECT * FROM Student WHERE Sname LIKE '李_'` 李白可以，李铁柱不行
    - 剩余示例不再赘述，可以拿着实验报告慢慢看

查询条件	谓词
比较	`=, >, <, >=, <=, !=,<>, !>, !<;` `NOT` 加上述比较运算符
确定范围	`BETWEEN AND, NOT BETWEEN AND`
确定集合	`IN, NOT IN`
字符匹配	`LIKE, NOT LIKE`
空值判断	`IS NULL, IS NOT NULL` *(重要)*
逻辑运算(多重条件)	`AND, OR, NOT`

对查询结果排序

- 用户可以使用 `ORDER BY` 子句对查询结果排序。可以按照一个或多个属性列的升序 (ASC) 或降序 (DESC) 排列，默认值为升序。
- 对于空值的情况，各个系统实现有所不同，但保持一致就行。

```
/*查询选了3号课程的学生成绩，按照分数降序排列*/
SELECT Sno,Grade
FROM SC
```

```
WHERE Cno='3'
ORDER BY Grade DESC
```

聚集函数

- 聚集函数用于统计列中的数据，仅可附加于 SELECT 语句后(返回统计结果)或 GROUP BY 的 HAVING 子句后(返回统计结果符合一定条件的数据组)
  - 若附加于SELECT语句后，此时它将作为一列数据返回，可以为它设置别名。
  - 当和聚集函数遇到空值时，除了 COUNT(*) 外，都跳过空值，只处理非空值。
  - 注意： WHERE 子句不能用聚集函数作为条件表达式。
- 常见的聚集函数见下表

聚集函数	作用
COUNT(*)	统计元组个数
COUNT([DISTINCT | ALL] <列名>)	统计一列中值的个数
SUM([DISTINCT | ALL] <列名>)	计算一列值的总和
AVG([DISTINCT | ALL] <列名>)	计算一列值的平均值
MAX([DISTINCT | ALL] <列名>)	求一列值的最大值
MIN([DISTINCT | ALL] <列名>)	求一列值的最小值

对查询结果分组

- 使用 GROUP BY 子句可以使查询结果按指定的一列或多列值分组，值相等的为一组。
  - 目的：细化聚集函数的作用对象
  - 使用 GROUP BY 子句分组后， SELECT 后的列名列表只能出现分组(作为分组标准)的列。
- 在分组之后，还可以使用 HAVING 短语指定条件，对组进行筛选。
  - 与 WHERE 子句的区别：
    - WHERE 子句作用于基本表或视图，从中选择满足条件的元组，即作用于每一条数据。 WHERE 后不能使用聚集函数，也是因为这一性质。(聚集函数的作用对象也是一组数据)
    - HAVING 短语作用于组，从中选择满足条件的组。

```
/*查询选修了三门以上课程学生的学号*/
SELECT Sno
FROM SC
GROUP BY Sno /*按照学生编号对成绩条目分组*/
HAVING COUNT(*)>3 /*从中筛选出包含元组数目大于3的组*/
```

3.3.2 连接查询

等值与非等值连接查询

- 同时查询多个表，只需在 `FROM` 后面增加需要查询的表。但如果没有 `WHERE` 子句限制条件，查询结果是这两个表的笛卡尔积。
- `WHERE` 子句可以规定两个及以上的表进行连接的条件（称为连接条件或连接谓词），同时查询这些表中的数据。
- 格式：`[<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>`
  - 比较运算符主要有 `=, >, <, >=, <= ,!=`
  - 当比较运算符为 `=` 时，称为等值连接，用其他运算符则为非等值连接。
  - 若在等值连接中，把目标列中重复的属性列去掉，则为自然连接。

```
SELECT Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade
FROM Student,SC
WHERE Student.Sno=SC.Sno;
```

自身连接

- 连接操作也可以在一个表和其本身之间进行，称为自身连接。
- 为此要为这个表取两个别名，以明确语义。

```
/*查询每一门课的简介先修课*/
SELECT FIRST.Cno, SECOND.Cpno
FROM Course FIRST,Course SECOND
WHERE FIRST.Cpno=SECOND.Cno
```

外连接

- 以其中一个表为主体，若另一个表中没有与该表中元组所对应的记录，仍然将这一元组保留在结果关系中（悬浮元组），并在对应列中填入空值 `NULL`。

```
SELECT Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade
FROM Student LEFT OUTER JOIN SC ON (Student.Sno=SC.Sno);
/*
        也可以使用USING来去掉结果中的重复值：
        FROM Student LEFT OUTER JOIN SC USING(Sno)
*/
```

多表连接

- 简单，直接看示例

```
SELECT Student.Sno,Sname,Cname,Grade
FROM Student,SC,Course
```

3.3.3 嵌套查询

3.3.4 集合查询

3.4 数据更新

3.5 视图

视图的作用

- 视图可以简化用户的操作
- 视图能使用户以多种角度看待同一数据
- 视图对重构数据库提供了一定程度的逻辑独立性
- 视图能够对机密数据提供安全保护
- 适当利用视图可以更清晰表达查询