第三章 SQL

- SQL概述
  - 特点
    - 集数据查询、数据操纵、数据定义和数据控制功能于一体
    - 高度非过程化
    - 面向集合的操作方式
    - 一种语法结构、两种使用方式
      - 既是自含式语言，又是嵌入式语言
  - SQL对象与三级模式结构的对应关系
    - 外模式－－ 视图、部分基本表
      - 视图
        - 是从基本表或其他视图中导出的表
        - 本身**不独立存储**在数据库中，也就是说数据库中**只存放视图的定义**而不存放视图对应的数据，这些数据仍存放在导出视图的基本表中，因此**视图是一个虚表**
    - 模式 －－ 基本表
      - 基本表
        - 是本身独立存在的表，SQL中一个关系就对应一个表
          - 一个表可以带若干索引，索引也存放在存储文件中
    - 内模式－－ 存储文件
      - 存储文件的逻辑结构组成了关系数据库的内模式
      - 存储文件的物理文件结构是任意的，对用户透明
- 学生-课程数据库

  学生表
  - Student(Sno, Sname, Sgender, Sage, Sdept)

  课程表
  - Course(Cno, Cname, Cpno, Ccredit)

  选课表
  - SC(Sno, Cno, Grade)
- 数据定义
  - 数据定义的概念
    - 对模式、基本表、视图、索引的定义、删除和修改(9种组合，**不能修改模式、视图和索引**)
  - 定义模式 CREATE SCHEMA

- **CREATE SCHEMA** schema name **AUTHORIZATION** username [schema_element [...]]

- 删除模式 DROP SCHEMA
  - **DROP SCHEMA** schema name <CASCADE | RESTRICT>
    - CASCADE和RESTRICT必须两者选其一
      - CASCADE级联
        - 表示在删除模式的同时把该模式中所有的数据库对象全部删除
      - RESTRICT限制
        - 表示如果该模式中已经定义了下属的数据库对象（eg，表，视图），则拒绝该删除语句的执行
        - 只有当该schema中没有任何下属的对象才能执行DROP SCHEMA
- 定义基本表CREATE TABLE
  - **CREATE TABLE** <表名> （<列名1><数据类型>[列级完整性约束条件] ,<列名2><数据类型>[列级完整性约束条件]...,[表级完整性约束条件]);
    - eg:Sno是主码，Sname取唯一值

```
CREATE TABLE student
(
   sno char(5) PRIMARY KEY,
   sname varchar(20) UNIQUE,
   sgender char(1),
   sage int,
   sdept char(15)
);
```

```
CREATE TABLE SC(
   sno char(5),
   cno char(5),
   grade int,
   PRIMARY KEY (sno,cno),
   FOREIGN KEY (sno) REFERENCES student,
   FOREIGN KEY (cno) REFERENCES course
);
```

    - UNIQUE
    - NOT NULL
    - CHECK

        - FOREIGN KEY(attribute_name1) REFERENCES table_name(attribute_name2)
            - 参照表和被参照表可以是同一个表
- 修改基本表 ALTER
    - 可增加新列，删除列上的完整性约束，修改列名及数据类型
    - **ALTER TABLE**[ADD[COLUMN] <new column_name><type[完整性约束]>],[ADD<表级完整性约束>]，[DROP[COLUMN]<列名> [CASCADE | RESTRICT]],[DROP CONSTRAINT<完整性约束名> [CASCADE | RESTRICT]] ,[ALTER COLUMN <列名> <type>]
        - 删除列
            - DROP [COLUMN] column[RESTRICT | CASCADE]
        - 增加表级完整性约束
            - ADD table_constraint
        - 列更名
            - ALTER TABLE name
            - RENAME [COLUMN] column TO new_column
    - 解决多个表间互相引用的问题（ppt32）
- 删除基本表 DROP TABLE
    - DROP TABLE <表名>;
- 建立和删除索引
    - 作用
        - 提高查询速度。
        - 如从O(n)到O(log2n)
    - 常需要建立索引的属性
        - 常出现在查询条件中
        - 常作为连接属性
    - 建立索引
        - CREATE [UNIQUE][CLUSTERED|NONCLUSTERED] INDEX <索引名> ON<表名>(<列名>[<次序>,[...]]);
        - UNIQUE(单一索引)：
            - 唯一索引，不允许存在索引值相同的两行
        - CLUSTERED(聚集索引)：
            - 索引项的顺序与表中记录的物理顺序一致。表中如果有多个记录在索引字段上相同，这些记录构成一簇，只有一个索引值。
            - 缺点：维护成本高，且一个表只能建一个聚簇索引。
            - 优点：查询速度快。
        - NONCLUSTERED(非聚集索引)
            - 作为非聚集索引，行的物理排序独立于索引排序

        - 非聚集索引的叶级包含索引行（B＋树）
    - 删除索引
        - DROP INDEX <索引名>;
- **SQL查询（重点掌握；难的是2-3层的嵌套查询）**
    - SQL查询一般格式

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]];

        - 含义
            - 根据**WHERE**子句的条件表达式，从**FROM**子句指定的基本表或视图中找出满足条件的元组，再按**SELECT**子句中的目标列表达式，选出元组中的属性值形成结果表
        - GROUP BY
            - 将结果按<列名1>的值进行分组，该属性值相等的元组为一个组
            - 通常会在组中作用**聚集函数**
            - 带HAVING，那么只有满足制定条件的组才能输出
        - ORDER BY
            - 将结果表按<列名2>的值升序或者降序排序
        - 简单的SQL查询与关系代数的联系
            - 简单的SQL查询与关系代数的联系：

SELECT A1,A2,…,An
FROM T1,T2,..,Tk
WHERE F;

相当于：

$$\Pi_{A1,A2,\ldots,An}\,\delta_F(T1\times T2\times\ldots\times Tk)$$

其中F中有的可能是连接条件，与后面的广义笛卡儿集构成连接。

    - 单表查询（比较简单）
        - 选择表中的若干列
            - 1. 查询指定列
                - eg：SELECT Sno,Sname FROM Student;
            - 2. 查询全部列
                - eg：SELECT * FROM Student;
                    - *，表示全部列

- 3. 查询经过计算的值（广义投影）
  - 改变列标题（SQL标准用AS关键字）
    - eg:

```
SELECT Sname AS Name, 'Year of Birth:' AS Birth,
  2004-Sage AS BirthYear, LOWER(Sdept) AS
  Department
  FROM Student;
```

- 选择表中的若干元组
- ORDER BY子句（查询结果排序）
  - 默认为升序(ASC)
  - **NULL值**在排序时被当作最大值

    注：这里说“NULL值最大”，仅仅针对NULL值排序的情况。
    如果取“NULL值”的字段出现在条件表达式中，将使条件计算为NULL，进而被排除于结果外。
  - 使用集函数
    - **COUNT**([DISTINCT|ALL]*) 统计元组个数
      - 除count(*)外，NULL值均被聚集函数所忽略
    - **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（此列必须是数值型）
    - **AVG**([DISTINCT|ALL] <列名>) 计算一列值的平均值（此列必须是数值型）
    - **MAX(**[DISTINCT|ALL] <列名>) 求一列值中的最大值
    - **MIN(**[DISTINCT|ALL] <列名>) 求一列值中的最小值
  - 对查询结果分组 GROUP BY
    - GROUP BY子句，可以将查询结果表的各行，按一列或多列取值相等的原则进行分组。
    - 对查询结果分组的目的
      - 是为了细化集函数的作用对象。
      - 如果未对查询结果分组，集函数将作用于整个查询结果，即整个查询结果只有一个函数值。否则，集函数将作用于每一个组，即每一组都有一个函数值
    - SQL规定，所有带有NULL值的记录在分组时被作为一组
    - 分组后，一些详细信息可能损失，不能出现在SELECT结果中
- 连接查询
- 嵌套查询
  - **查询块**
    - 一个SELECT-FROM-WHERE语句
  - **嵌套查询 nested query**

- 将一个查询块嵌套在另一个查询块的**WHERE子句**或**HAVING短语**的条件中的查询
- **外层查询或父查询**
- **内层查询或子查询**
- 通常，嵌套查询的求解方法是**由里向外处理**
- 查询的SELECT语句中不能使用ORDER BY子句
  - order by只能对最终的查询结果进行排序
- 嵌套查询类型
  - **1. 带有IN谓词的子查询**
    - 是指父查询与子查询之间用IN进行连接，判断某个属性列值是否在子查询的结果中
    - 例子ppt115
    - 不相关子查询
      - 子查询的查询条件不依赖于父查询
    - 相关子查询
      - 子查询条件依赖于父查询
      - 整个语句称为相关嵌套查询
  - **2.带有比较运算符的子查询**
    - 是指父查询与子查询之间用比较运算符进行连接
    - 当用户能确切知道内层查询返回的是单值时，可以用>、<、=、>=、<=、!=或<>等比较运算符
    - x是sc的别名，这种查询就是**相关子查询**，y.sno=x.sno限制了查询哪位学生的平均成绩

      例3.57 找出每个学生超过他选修课程平均成绩的课程号。

      ```
      SELECT Sno, Cno
      FROM SC x
      WHERE Grade>=(SELECT AVG(Grade)
                    FROM SC y
                    WHERE y.Sno=x.Sno);
      ```

      查这样的选课

      该选课成绩比选课学生的平均成绩要高
    - 相关子查询的一般求解过程
      - 外层每一元组都被带入内层。
      - 内层得到结果。
      - 外层WHERE条件判断。如果为真，当前元组被选出；否则，不被选出。
      - 重复，直到外层没有元组为止。

- **3. 带有ANY（SOME）或ALL谓词的子查询**
  - 比较运算符一般不能与集合比较。集合前加以谓词ANY（SOME）或ALL修饰后可以比较。
    - 子查询返回单值：用比较运算符；子查询返回多值：用ANY（SOME）或ALL谓词修饰符
    - 使用ANY（SOME）或ALL谓词时则必须同步使用比较运算符
    - （any）Some R 意义为 Some in R。R中的某一个
    - All R 意义为 all in R, 或 all of R。R中的全部
  - != equal to <>
  - 可以用聚集函数（max，min，...）,聚集函数实现子查询通常比直接用ALL或ANY查询效率要高
  - ALL，ANY与聚集函数的对应关系（书P109）
- **4. 带有EXISTS谓词的子查询**
  - **EXISTS**代表存在量词∃
  - 带有EXISTS谓词的子查询**不返回任何实际数据**，它只产生逻辑真值“true”或逻辑假值“false”
  - 常用于判断**x∈R，S R，S=R，S∩R非空等**
  - 通常是相关子查询
  - ppt136
    - 由于是EXIST引出的子查询，为目标列表达式常用*，因为带EXIST的子查询只返回真值或假值，给出列名无意义
  - 使用存在量词**NOT EXIST**后，若内层查询结果为空，则外层的WHERE子句返回真值，否则返回假值
  - 一些带EXISTS或NOT EXISTS谓词的子查询不能被其他形式的子查询等价替换；但所有带IN谓词、比较运算符、ANY和ALL谓词的子查询都能用带EXISTS谓词的子查询等价替换。
  - SQL语言中没有全称量词（For all），因此可以利用谓词演算将一个**带有全称量词的谓词**转换为等价的带有**存在量词的谓词**
    - ppt140
  - **not exists** *r equal to r = Ø*
  - **exists** *r equal to r != Ø*
  - *用于检查包含关系*
    - A包含B
      - ==B-A=Ø
      - 等于not exist(B except A)
      - 例子ppt149，书111
- 集合查询

        - UNION 并
        - INTERSECT 交
        - EXCEPT 差
        - 参加集合操作的各查询结果列数必须相同，对应项的数据类型也必须相同
    - 基于派生表的查询
    - 书114
- 数据更新
    - 插入数据 INSERT
        - 插入单个元组

```
INSERT
    INTO <表名> [(<属性列1>[,<属性列2>...)]
    VALUES (<常量1> [,<常量2>]...)
```

            - if 某些属性列在INTO子句中没有出现
                - 新记录在这些列上将取空值。
                - 在表定义时说明为NOT NULL的属性列不能取空值。
                - 主码属性不能取空值
            - if INTO子句中没有指明任何列名
                - 新插入的记录必须在每个属性列上均有值。
                - VALUES子句后值与属性列的顺序一致。
            - eg：ppt156
        - 插入子查询结果
            - 子查**询嵌套在INSERT语句**中，用以生成要插入的数据
            - 插入子查询结果的INSERT语句的格式

```
INSERT
    INTO <表名> [(<属性列1> [,<属性列2>...)]
    子查询;
```

            - 可以实现批量插入
                - 一次将子查询的结果全部插入指定表中
            - eg：ppt159
    - 修改数据 UPDATE
        - 更新操作
            - UPDATE语句
            - 格式

```
UPDATE <表名>
SET <列名>=<表达式>[,<列名>=<表达式>]...
[WHERE <条件>];
```

- 其功能是**修改指定表**中**满足WHERE子句条件的**元组
  - SET子句用于指定修改方法，即用<表达式>的值取代相应的属性列值
- eg：ppt162

- 带子查询的修改语句
  - 子查询也可以嵌套在UPDATE语句中，用以构造执行修改操作的条件
  - eg：ppt163

- 删除数据 DELETE
  - 删除元组

    - DELETE语句的功能是从指定表中删除满足WHERE子句条件的所有元组
    - DELETE语句删除的是表中的数据，而不是关于表的定义（drop）
  - 删除多个元组的值
    - DELETE FROM SC;
      - 这条DELETE语句将使SC成为空表，它删除了SC的所有元组
  - 带子查询的删除语句
    - 子查询同样也可以嵌套在DELETE语句中，用以构造执行删除操作的条件
    - ppt169
- 对某个基本表中的数据进行增删改操作可能会破坏**参照完整性**

- 视图
  - 视图：
    - 定义：
      - 从一个或多个基本表(或视图)导出的表
      - （是一个虚表）
    - 功能：
      - 使用户以不同方式看数据；
    - 主要目的：

        - 提供数据库保护
    - 数据库中只存视图的定义，不存数据；
    - 对视图的更新有限制。
    - 视图的优点
        - 视图能够**简化**用户的**操作**
        - 视图使用户能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
        - 合理利用视图，能方便复杂查询
- 定义视图 **CREATEVIEW**
    - 创建视图 **CREATEVIEW**
        - 格式

          ```
          CREATE VIEW <视图名>[(<列名>[,<列名>]...)]
          AS <子查询>
          [WITH CHECK OPTION];
          ```

        - 行列子集视图
            - 从一个基本表中导出，只是去掉了某些行或列(保留原表的主码)，这样的视图称为行列子集视图
        - 带表达式的视图
            - 即带虚拟列的视图。
        - 分组视图
            - 子查询带集函数和GROUPBY分组的视图
- 删除视图 **DROP VIEW**
    - 格式

      ```
      DROP VIEW <视图名> [CASCADE|RESTRICT];
      ```

        - CASCADE 级联删除
        - RESTRICT 受限删除 默认
    - 不会影响定义视图的基本表的数据和定义
- 查询视图
    - DBMS执行对视图的查询转换成对基本表的查询
    - 视图的消解（ViewResolution）
        - 将对视图的查询转换为对基本表的查询的过程
    - 现代DBMS对视图的消解一般不会出错
        - 在应用中，对基本表的查询和对视图的查询可以不做区别
- 更新视图
    - 对视图的更新，最终要转换为对基本表的更新

        - 为防止用户通过视图对数据进行增删改时，无意或故意操作不属于视图范围内的基本表数据，可在定义视图时加上**WITH CHECK OPTION**子句
        - 行列子集视图是可更新的
- 嵌入式SQL
    - SQL语言使用方式
        - 在终端交互式方式下使用（独立语言）
        - 将SQL语言嵌入到某种高级语言如Pascal、COBOL、FORTRAN、C中使用（嵌入式SQL）
            - 将SQL嵌入到高级语言中混合编程，SQL语句负责操纵数据库
            - 高级语言语句负责控制程序流程
    - 一般形式
        - 语句级接口
            - 如PB Script。
        - 调用级接口
            - 如JDBCODBC
    - 最重要的是两者之间的通信问题：
        - 1 向主语言传递SQL语句的执行状态；
            - SQL语句执行后，系统要反馈给应用程序若干信息，主要包括描述系统当前工作状态和运行环境的各种数据，这些信息将送到SQL通信区SQLCA中。应用程序从SQLCA中取出这些状态信息，据此决定接下来执行的语句
        - 游标
            - 为处理集合结果而引入。
        - 2 主语言向SQL提供参数(主变量实现)；
        - 3 SQL查询结果交还主语言处理
    -

以上内容整理于 幕布文档