数据库技术与应用

北京邮电大学计算机学院 肖达

xiaoda99@gmail.com

第三章 关系数据库标准语言SQL

SQL概述

数据查询

数据定义

数据更新

视图

SQL概述

- SQL（Structured Query Language）

  结构化查询语言，是关系数据库的标准语言

- SQL是一个通用的、功能极强的关系数据库语言

SQL的产生和发展

- 1974年，由Boyce和Chamberlin提出
- 1975~1979，IBM San Jose Research Lab的关系数据库管理系统原型System R实施了这种语言.该语言最初叫做SQUEL(Structured English Query Language)
- ANSI发布的SQL-86是第一个SQL标准
- SQL-89、SQL-92(SQL2)、SQL-99(SQL3)
- 现状：大部分DBMS产品都支持SQL，成为操作数据库的标准语言，但支持程度不同

SQL的特点

1.综合统一

- 集数据定义语言（DDL），数据操纵语言（DML），数据控制语言（DCL）功能于一体。
- 可以独立完成数据库生命周期中的全部活动：
  - 定义关系模式，插入数据，建立数据库；
  - 对数据库中的数据进行查询和更新；
  - 数据库重构和维护
  - 数据库安全性、完整性控制等
- 用户数据库投入运行后，可根据需要随时逐步修改模式，不影响数据的运行。
- 数据操作符统一

2.高度非过程化

- 非关系数据模型的数据操纵语言“面向过程”，必须指定存取路径
- SQL只要提出“做什么”，无须了解存取路径。
- 存取路径的选择以及SQL的操作过程由系统自动完成。

3.面向集合的操作方式

- 非关系数据模型采用面向记录的操作方式，操作对象是一条记录
- SQL采用集合操作方式
  - 操作对象、查找结果可以是元组的集合
  - 一次插入、删除、更新操作的对象可以是元组的集合

4.多种使用方式

- 交互式SQL
  - 一般DBMS都提供联机交互工具
  - 用户可直接键入SQL命令对数据库进行操作
  - 由DBMS来进行解释
- 嵌入式SQL
  - 能将SQL语句嵌入到高级语言（宿主语言）
  - 使应用程序充分利用SQL访问数据库的能力、宿主语言的过程处理能力
  - 一般需要预编译，将嵌入的SQL语句转化为宿主语言编译器能处理的语句

5.语言简洁，易学易用

- SQL功能极强，完成核心功能只用了9个动词。

表 3.1　SQL 语言的动词

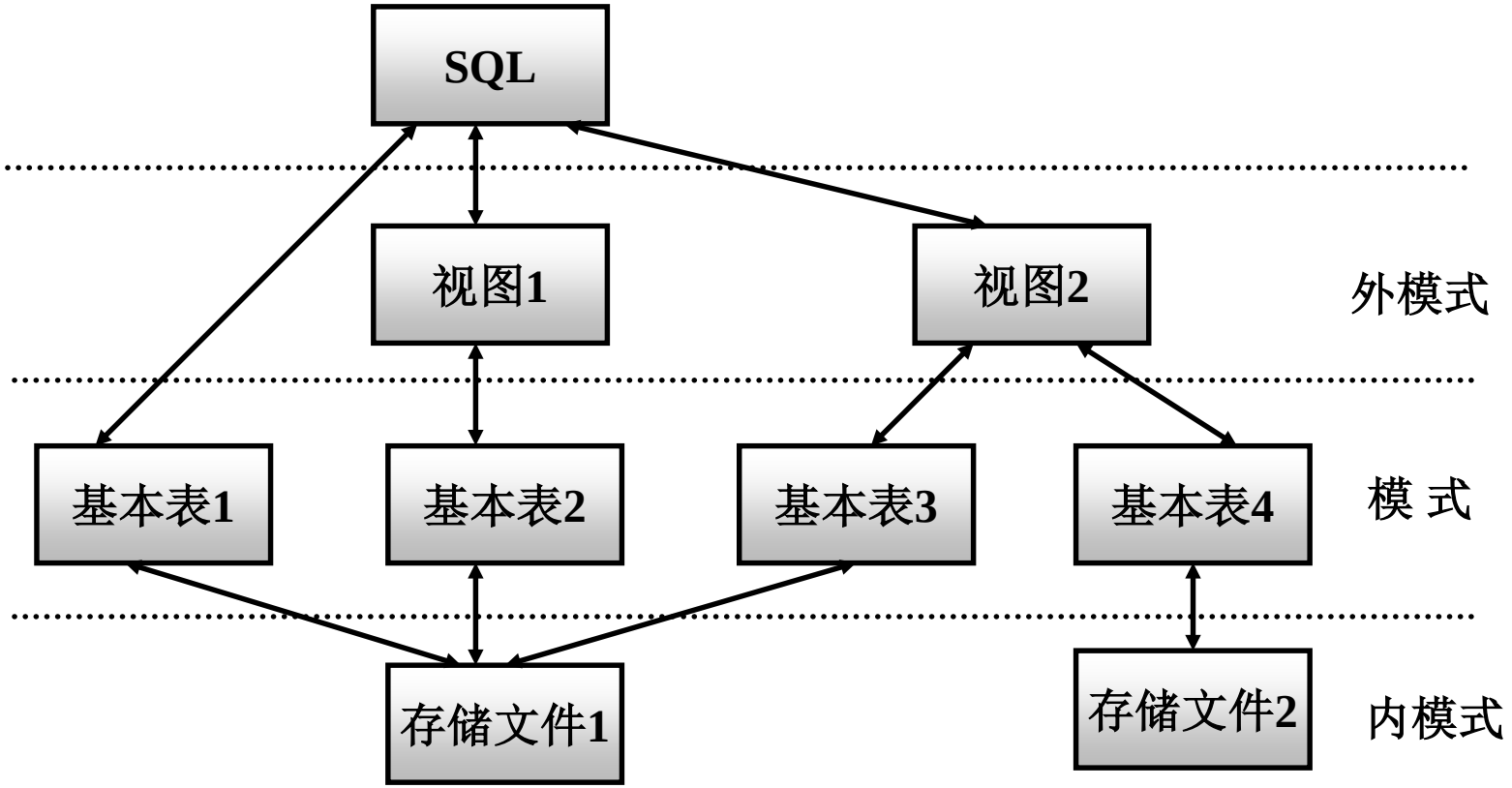
SQL 功能	动　词
数据查询	SELECT
数据定义	CREATE，DROP，ALTER
数据操纵	INSERT，UPDATE，DELETE
数据控制	GRANT，REVOKE

SQL的功能

- 数据定义（DDL）
  - 定义、删除、修改基本表（Base Table）
  - 定义、删除视图（View）
  - 定义、删除索引（Index）
- 数据操纵（DML）
  - 数据查询
  - 数据增、删、改
- 数据控制（DCL）
  - 用户访问权限的授予、收回

SQL数据库的体系结构

SQL支持关系数据库三级模式结构



SQL的基本概念

- 基本表
  - 本身独立存在的表
  - SQL中一个关系就对应一个基本表
  - 一个(或多个)基本表对应一个存储文件
  - 一个表可以带若干索引
- 存储文件
  - 逻辑结构组成了关系数据库的内模式
  - 物理结构是任意的，对用户透明
- 视图
  - 从一个或几个基本表导出的表
  - 数据库中只存放视图的定义而不存放视图对应的数据
  - 视图是一个虚表
  - 用户可以在视图上再定义视图

第三章 关系数据库标准语言SQL

SQL概述

数据查询

数据定义

数据更新

视图

数据查询

- 数据查询是数据库应用的核心功能
- 一个典型的SQL查询表达式的基本结构为
  - select 属性名表
  - from 关系名表
  - where (条件表达式)

$$\pi_{A_1, A_2, \ldots, A_n}(\sigma_p(r_1 \times r_1 \times \ldots \times r_m))$$

Select Where From

SQL查询表达式的含义

- 对 From 子句中的各关系，作笛卡尔积（×）
- 对 Where 子句中的逻辑表达式进行选择（σ）运算，找出符合条件的元组
- 根据 Select 子句中的属性列表，对上述结果作投影（π）操作
- 结果集：查询操作的对象是关系，结果也是一个关系，是一个动态数据集。

数据查询

- 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]];

数据查询

- 单表查询
- 集合查询
- 连接查询
- 嵌套查询
- **Select语句的一般形式**

学生-课程 数据库

- 学生-课程模式 S-T：

  学生表：Student(Sno,Sname,Ssex,Sage,Sdept)

  课程表：Course(Cno,Cname,Cpno,Ccredit)

  学生选课表：SC(Sno,Cno,Grade)

一、选择表中的若干列

- 1、查询指定列

[例] 查询全体学生的学号与姓名。

```
SELECT Sno，Sname
FROM Student；
```

[例] 查询全体学生的姓名、学号、所在系。

```
SELECT Sname，Sno，Sdept
FROM Student；
```

2. 查询全部列

- 选出所有属性列：
  - 在SELECT关键字后面列出所有列名
  - 将<目标列表达式>指定为 *

[例] 查询全体学生的详细记录。

```
SELECT Sno，Sname，Ssex，Sage，Sdept
FROM Student；
```

或

```
SELECT *
FROM Student；
```

3. 查询经过计算的值

- SELECT子句的<目标列表达式>可以为：
  - 算术表达式
  - 字符串常量
  - 函数
  - 列别名

查询经过计算的值（续）

[例] 查全体学生的姓名及其出生年份。

```
SELECT Sname，2009-Sage    /*假定当年的年份为2009年
   */
FROM Student；
```

输出结果：

Sname	2009-Sage
李勇	1987
刘晨	1988
王敏	1989
张立	1988

查询经过计算的值（续）

[例] 查询全体学生的姓名、出生年份和所有系，要求用小写字母表示所有系名

```
SELECT Sname, 'Year of Birth: ', 2009-Sage,
  LOWER(Sdept)
FROM Student;
```

输出结果：

Sname	'Year of Birth:'	2009-Sage	LOWER(Sdept)
李勇	Year of Birth:	1987	cs
刘晨	Year of Birth:	1988	is
王敏	Year of Birth:	1989	ma
张立	Year of Birth:	1988	is

查询经过计算的值（续）

- 使用列别名改变查询结果的列标题:

SELECT Sname NAME，'Year of Birth: ′ BIRTH，
2009-Sage BIRTHDAY，LOWER(Sdept) DEPARTMENT
FROM Student；

输出结果：

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth:	1987	cs
刘晨	Year of Birth:	1988	is
王敏	Year of Birth:	1989	ma
张立	Year of Birth:	1988	is

单表查询

- 查询仅涉及一个表：
  - 一、 选择表中的若干列
  - 二、 选择表中的若干元组
  - 三、 ORDER BY子句
  - 四、 聚集函数
  - 五、 GROUP BY子句

二、选择表中的若干元组

- 1. 消除取值重复的行

  如果没有指定DISTINCT关键词，则缺省为ALL

  [例] 查询选修了课程的学生学号。

  SELECT Sno FROM SC；

  等价于：

  SELECT ALL Sno FROM SC；

  执行上面的SELECT语句后，结果为：

Sno
200215121
200215121
200215121
200215122
200215122

消除取值重复的行（续）

- 指定DISTINCT关键词，去掉表中重复的行

```
SELECT DISTINCT Sno
FROM SC;
```

执行结果：

Sno
200215121
200215122

2.查询满足条件的元组

表3.4 常用的查询条件

查 询 条 件	谓　词
比　较	=，>，<，>=，<=，!=，<>，!>，!<；NOT+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	IS NULL，IS NOT NULL
多重条件（逻辑运算）	AND，OR，NOT

(1) 比较大小

[例] 查询计算机科学系全体学生的名单。

```
SELECT Sname
FROM Student
WHERE Sdept='CS';
```

[例] 查询所有年龄在20岁以下的学生姓名及其年龄。

```
SELECT Sname，Sage
 FROM    Student
 WHERE Sage < 20;
```

[例] 查询考试成绩有不及格的学生的学号。

```
SELECT DISTINCT Sno
FROM  SC
WHERE Grade<60;
```

(2) 确定范围

- **谓词：** BETWEEN … AND …

  NOT BETWEEN … AND …

[例] 查询年龄在20~23岁（包括20岁和23岁）之间的学生的姓名、系别和年龄

```
SELECT Sname，Sdept，Sage
FROM    Student
WHERE   Sage BETWEEN 20 AND 23；
```

[例] 查询年龄不在20~23岁之间的学生姓名、系别和年龄

```
SELECT Sname，Sdept，Sage
FROM    Student
WHERE Sage NOT BETWEEN 20 AND 23；
```

(3) 确定集合

- **谓词：IN <值表>，NOT IN <值表>**

[例]查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别。

```
SELECT Sname，Ssex
FROM  Student
WHERE Sdept IN ( 'IS'，'MA'，'CS' );
```

[例]查询既不是信息系、数学系，也不是计算机科学系的学生的姓名和性别。

```
SELECT Sname，Ssex
FROM Student
WHERE Sdept NOT IN ( 'IS'，'MA'，'CS' );
```

(4)字符匹配

- **谓词：** [NOT] LIKE ‘<匹配串>’ [ESCAPE ‘ <换码字符>’]

1) 匹配串为固定字符串

[例] 查询学号为200215121的学生的详细情况。

```
SELECT *
 FROM  Student
 WHERE  Sno LIKE ‘200215121';
```

等价于：

```
SELECT  *
 FROM  Student
 WHERE Sno = '200215121';
```

字符匹配（续）

2) 匹配串为含通配符的字符串

[例] 查询所有姓刘学生的姓名、学号和性别。

```
SELECT Sname，Sno，Ssex
FROM Student
WHERE  Sname LIKE ‘刘%’；
```

[例] 查询姓"欧阳"且全名为三个汉字的学生的姓名。

```
SELECT Sname
FROM   Student
WHERE  Sname LIKE '欧阳__';
```

字符匹配（续）

[例] 查询名字中第2个字为"阳"字的学生的姓名和学号。

```
SELECT Sname，Sno
FROM Student
WHERE Sname LIKE ‘__阳%’；
```

[例] 查询所有不姓刘的学生姓名。

```
SELECT Sname，Sno，Ssex
FROM Student
WHERE Sname NOT LIKE '刘%'；
```

字符匹配（续）

3) 使用换码字符将通配符转义为普通字符

[例] 查询DB_Design课程的课程号和学分。

```
SELECT Cno，Ccredit
FROM Course
WHERE Cname LIKE 'DB\_Design' ESCAPE '\‘；
```

[例] 查询以"DB_"开头，且倒数第3个字符为 i的课程的详细情况。

```
SELECT  *
FROM   Course
WHERE  Cname LIKE  'DB\_%i_ _' ESCAPE ' \ ‘；
```

ESCAPE '\' 表示“ \ ” 为换码字符

(5) 涉及空值的查询

- **谓词：** IS NULL 或 IS NOT NULL
- “IS” 不能用 “=” 代替

[例] 某些学生选修课程后没有参加考试，所以有选课记录，但没有考试成绩。查询缺少成绩的学生的学号和相应的课程号。

```
SELECT Sno，Cno
 FROM  SC
 WHERE  Grade IS NULL
```

[例] 查所有有成绩的学生学号和课程号。

```
SELECT Sno，Cno
FROM  SC
WHERE  Grade IS NOT NULL；
```

(6) 多重条件查询

- 逻辑运算符：AND和 OR来联结多个查询条件
  - AND的优先级高于OR
  - 可以用括号改变优先级
- 可用来实现多种其他谓词
  - [NOT] IN
  - [NOT] BETWEEN … AND …

[例] 查询计算机系年龄在20岁以下的学生姓名。

```
SELECT Sname
FROM  Student
WHERE Sdept= 'CS' AND Sage<20;
```

多重条件查询（续）

- **改写**

[例] 查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别。

```
SELECT Sname，Ssex
FROM Student
WHERE Sdept IN ( 'IS'，'MA'，'CS' )
```

可改写为：

```
SELECT Sname，Ssex
FROM   Student
WHERE  Sdept= ' IS ' OR Sdept= ' MA' OR Sdept= ' CS ';
```

例子：银行客户管理

分行表：*branch* (*branch-name, branch-city, assets)*

客户表：*customer* (*customer-name, customer-street, customer-city)*

存款账户表： *account* (*account-number, branch-name, balance)*

贷款账户表： *loan* (*loan-number, branch-name, amount)*

存款人表： *depositor* (*customer-name, account-number)*

贷款人表： *borrower* (*customer-name, loan-number)*

- *branch (branch-name, branch-city, assets)*
- *customer (customer-name, customer-street, customer-city)*
- *account (account-number, branch-name, balance)*
- *loan (loan-number, branch-name, amount)*
- *depositor (customer-name, account-number)*
- *borrower (customer-name, loan-number)*

- 贷款额在90000－100000之间的贷款账号。

```
select  loan-number
from   loan
where  amount  between 90000 and  100000
```

- 列出居住在Rye, Stamford, Harrison三个城市的客户名字。

```
select  customer-name
from    customer
where  customer-city in ('Rye', 'Stamford', 'Harrison')
```

- 找出在Perridge分行贷款而且贷款额多于1300的贷款号。

```
select     loan-number
from       loan
where    branch-name='Perridge' and  amount>1300
```

单表查询

- 查询仅涉及一个表：
  - 一、 选择表中的若干列
  - 二、 选择表中的若干元组
  - 三、 ORDER BY子句
  - 四、 聚集函数
  - 五、 GROUP BY子句

三、ORDER BY子句

- ORDER BY子句
  - 可以按一个或多个属性列排序
  - 升序：ASC；降序：DESC；缺省值为升序
- 当排序列含空值时
  - ASC：排序列为空值的元组最后显示
  - DESC：排序列为空值的元组最先显示

ORDER BY子句（续）

[例] 查询选修了3号课程的学生的学号及其成绩，查询结果按分数降序排列。

```
SELECT Sno，Grade
FROM  SC
WHERE  Cno= ' 3 '
ORDER BY Grade DESC；
```

[例] 查询全体学生情况，查询结果按所在系的系号升序排列，同一系中的学生按年龄降序排列。

```
SELECT  *
FROM  Student
ORDER BY Sdept，Sage DESC；
```

单表查询

- 查询仅涉及一个表：
  - 一、 选择表中的若干列
  - 二、 选择表中的若干元组
  - 三、 ORDER BY子句
  - 四、 聚集函数
  - 五、 GROUP BY子句

四、聚集函数

- 聚集函数：
  - 计数

    COUNT（[DISTINCT | ALL] *）

    COUNT（[DISTINCT | ALL] <列名>）
  - 计算总和

    SUM（[DISTINCT | ALL] <列名>）
  - 计算平均值

    AVG（[DISTINCT | ALL] <列名>）
  - 最大最小值

    MAX（[DISTINCT | ALL] <列名>）

    MIN（[DISTINCT | ALL] <列名>）

聚集函数（续）

[例] 查询学生总人数。

```
SELECT COUNT(*)
FROM  Student；
```

[例] 查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT Sno)
FROM SC；
```

[例] 计算1号课程的学生平均成绩。

```
SELECT AVG(Grade)
FROM SC
WHERE Cno= ' 1 '；
```

聚集函数（续）

[例] 查询选修1号课程的学生最高分数。

```
SELECT MAX(Grade)
FROM SC
WHER Cno=‘1’;
```

［例］查询学生200215012选修课程的总学分数。

```
SELECT SUM(Ccredit)
FROM  SC，  Course
WHER Sno='200215012' AND SC.Cno=Course.Cno;
```

单表查询

- 查询仅涉及一个表：
  - 一、 选择表中的若干列
  - 二、 选择表中的若干元组
  - 三、 ORDER BY子句
  - 四、 聚集函数
  - 五、 GROUP BY子句

五、GROUP BY子句

- GROUP BY子句分组：

  细化聚集函数的作用对象

  - 未对查询结果分组，聚集函数将作用于整个查询结果
  - 对查询结果分组后，聚集函数将分别作用于每个组
  - 作用对象是查询的中间结果表
  - 按指定的一列或多列值分组，值相等的为一组

GROUP BY子句（续）

[例] 求各个课程号及相应的选课人数。

```
SELECT Cno，COUNT(Sno)
FROM   SC
GROUP BY Cno；
```

查询结果：

Cno	COUNT(Sno)
1	22
2	34
3	44
4	33
5	48

SC

Sno	Cno	Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

[例] 查询选修了3门以上课程的学生学号。

```
SELECT Sno
FROM  SC
GROUP BY Sno
HAVING  COUNT(*) >3；
```

GROUP BY子句（续）

- 查询平均成绩大于等于90分的学生学号和平均成绩
  下面的语句是否正确：
  SELECT Sno, AVG(Grade)
  FROM SC
  WHERE AVG(Grade)>=90
  GROUP BY Sno;
- 错误！因为**WHERE子句中是不能用聚集函数作为条件表达式**
- 正确的查询语句应该是：
- SELECT Sno, AVG(Grade)
  FROM SC
  GROUP BY Sno
  HAVING AVG(Grade)>=90;

GROUP BY子句（续）

- HAVING短语与WHERE子句的区别：
  - 作用对象不同
  - WHERE子句作用于基表或视图，从中选择满足条件的元组
  - HAVING短语作用于组，从中选择满足条件的组。

- *branch* (*branch-name*, *branch-city*, *assets*)
- *customer* (*customer-name*, *customer-street*, *customer-city*)
- *account* (*account-number*, *branch-name*, *balance*)
- *loan* (*loan-number*, *branch-name*, *amount*)
- 找出每个分行的存款帐户余额

```
select   branch-name, avg (balance)
from     account
group    by    branch-name
```

- 找出存款帐户平均余额大于1200的分行

```
select   branch-name
from     account
group    by    branch-name
having   avg  (balance)>1200
```

- 找出余额大于1,000,000的存款账户数大于10的分行

```
select   branch-name
from     account
where    balance > 1000000
group    by    branch-name
having   count(*) > 10
```