

第3章: 结构化查询语言

Structured Query Language (SQL)

邹兆年

哈尔滨工业大学
计算机科学与技术学院
海量数据计算研究中心
电子邮件: znzou@hit.edu.cn

2019年春

教学内容¹

- ① SQL数据定义
 - ▶ 基本数据类型
 - ▶ 创建关系模式
 - ▶ 修改关系模式
 - ▶ 删除关系模式
 - ▶ 定义视图
- ② SQL数据更新
 - ▶ 插入数据
 - ▶ 修改数据
 - ▶ 删除数据
 - ▶ 数据完整性检查
- ③ SQL数据查询
 - ▶ 单关系查询
 - ▶ 连接查询
 - ▶ 嵌套查询

¹课件更新于2019年3月27日

3.1 SQL数据定义

SQL Data Definition

数据类型

SQL提供的基本数据类型有: 数值型、日期时间型、串型、枚举型

数值型

- 整型

- ▶ INTEGER或INT: 4字节
- ▶ SMALLINT: 2字节
- ▶ TINYINT: 1字节
- ▶ MEDIUMINT: 3字节
- ▶ BIGINT: 8字节

- 定点型

- ▶ DECIMAL(p,s): 有效数字共 p 位, 小数点后 s 位
- ▶ NUMERIC(p,s): 同DECIMAL(p,s)

- 浮点型

- ▶ FLOAT: 单精度, 4字节
- ▶ DOUBLE: 双精度, 8字节

- 二进制数型

- ▶ BIT(b): b 位二进制数($1 \leq b \leq 64$), $(b+7)/8$ 字节, 输入时用 b '101'表示5

数据类型(续)

日期时间型

- **YEAR**: 年类型
 - ▶ 存储: 1字节整数YYYY
 - ▶ 显示和输入格式: 'YYYY'
- **DATE**: 日期型
 - ▶ 存储: 3字节整数 $YYYY \times 16 \times 32 + MM \times 32 + DD$
 - ▶ 显示和输入格式: 'YYYY-MM-DD'
- **TIME**: 时间型
 - ▶ 存储: 3字节整数 $DD \times 24 \times 3600 + HH \times 3600 + MM \times 60 + SS$
 - ▶ 显示和输入格式: 'HH:MM:SS' 或 'HHH:MM:SS' (为什么?)
- **DATETIME**: 日期时间型(与时区无关)
 - ▶ 存储: 8字节整数, 其中4字节存储整数YYYYMMDD, 另外4字节存储整数HHMMSS
 - ▶ 显示和输入格式: 'YYYY-MM-DD HH:MM:SS'
- **TIMESTAMP**: 时间戳型(和UNIX时间戳相同, 与时区相关)
 - ▶ 存储: 4字节整数, 保存从1970-01-01 00:00:00 UTC以来的秒数
 - ▶ 显示和输入格式: 'YYYY-MM-DD HH:MM:SS'

数据类型(续)

字符串型

- **CHAR(n)**: 定长字符串型, 最多存储 n 个字符($n \leq 255$)
 - ▶ 存储空间固定
 - ▶ 当字符串不足 n 个字符时, 后面用空格补全
- **VARCHAR(n)**: 变长字符串型, 最多存储 n 个字符($n \leq 65535$)
 - ▶ 存储空间根据字符串的实际字节长度 L 变化
- **TEXT**: 文本型
 - ▶ **TINYTEXT**: $L \leq 2^8$
 - ▶ **TEXT**: $L \leq 2^{16}$
 - ▶ **MEDIUMTEXT**: $L \leq 2^{24}$
 - ▶ **LONGTEXT**: $L \leq 2^{32}$

类型	所需存储空间(L : 字符串的实际字节长度)
CHAR(n)	nw , 其中 w 是字符的最大长度
VARCHAR(n)	如果 $0 \leq n \leq 255$, 则需 $L + 1$ 字节; 如果 $n > 255$, 则需 $L + 2$ 字节
TINYTEXT	$L + 1$ 字节
TEXT	$L + 2$ 字节
MEDIUMTEXT	$L + 3$ 字节
LONGTEXT	$L + 4$ 字节

数据类型(续)

二进制串型

- **BINARY(*n*)**: 定长二进制串型, 最多存储 n 个字节($n \leq 255$)
 - ▶ 存储空间固定, n 字节
 - ▶ 当二进制串不足 n 个字节时, 后面用\0补全
- **VARBINARY(*n*)**: 变长二进制串型, 最多存储 n 个字节($n \leq 65535$)
 - ▶ 存储空间根据二进制串实际字节长度 L 变化
- **BLOB**: 二进制对象型(Binary Large OBject)
 - ▶ **TINYBLOB**: $L \leq 2^8$
 - ▶ **BLOB**: $L \leq 2^{16}$
 - ▶ **MEDIUMBLOB**: $L \leq 2^{24}$
 - ▶ **LONGBLOB**: $L \leq 2^{32}$

类型	所需存储空间(L : 字符串的实际字节长度)
BINARY(n)	n 字节
VARBINARY(n)	如果 $0 \leq n \leq 255$, 则需 $L + 1$ 字节; 如果 $n > 255$, 则需 $L + 2$ 字节
TINYBLOB	$L + 1$ 字节
BLOB	$L + 2$ 字节
MEDIUMBLOB	$L + 3$ 字节
LONGBLOB	$L + 4$ 字节

数据类型(续)

枚举型

- **ENUM(值列表)**: 枚举型
 - ▶ 例: `ENUM('Mercury', 'Venus', 'Earth', 'Mars')`
 - ▶ 枚举型的值只能取自值列表
 - ▶ 值列表中最多包含65535个不同的值
 - ▶ 如果值列表中包含至多255个值, 则占1字节; 如果值列表中超过255个值, 则占2字节

集合型

- **SET(值列表)**: 集合型
 - ▶ 例: `SET('Mercury', 'Venus', 'Earth', 'Mars')`
 - ▶ 集合型的值只能是值集合的子集, 如`'Mercury,Earth'`
 - ▶ 值列表中最多包含64个不同的值
 - ▶ 存储为二进制数, 列表中每个值对应一个二进制位。值在子集中存在, 则相应位置为1; 否则, 置为0。例如, `'Mercury,Earth'`存储为0101
 - ▶ 占用空间大小取决于集合中元素的个数

创建关系模式

- **功能:** 定义关系模式, 包括关系名、属性名、属性类型、主键、外键、完整性约束等
- **SQL语句:** **CREATE TABLE**

Example (创建关系模式)

创建Student关系, 其模式如下:

属性名	属性类型	属性含义
Sno	CHAR(6)	学号
Sname	VARCHAR(10)	姓名
Ssex	ENUM('M', 'F')	性别
Sage	INT	年龄
Sdept	VARCHAR(20)	所在系

SQL语句:

```
CREATE TABLE Student (  
  Sno CHAR(6),  
  Sname VARCHAR(10),  
  Ssex ENUM('M', 'F'),  
  Sage INT,  
  Sdept VARCHAR(20)  
);
```

声明主键

- **功能:** 声明关系的主键(一个关系仅有一个主键), 有两种声明方法
- **SQL关键词:** **PRIMARY KEY**

Example (声明主键)

将Sno属性声明为Student关系的主键

方法1:

```
CREATE TABLE Student (  
  Sno CHAR(6) PRIMARY KEY,  
  Sname VARCHAR(10),  
  Ssex ENUM('M', 'F'),  
  Sage INT,  
  Sdept VARCHAR(20)  
);
```

方法2:

```
CREATE TABLE Student (  
  Sno CHAR(6),  
  Sname VARCHAR(10),  
  Ssex ENUM('M', 'F'),  
  Sage INT,  
  Sdept VARCHAR(20),  
  PRIMARY KEY (Sno)  
);
```

声明多个属性构成的主键时, 只能用第2种方法

声明外键

- 功能: 声明关系的外键(一个关系可以有多个外键)
- SQL关键词: **FOREIGN KEY**

Example (声明外键)

创建SC关系, 其模式如下:

属性名	属性类型	属性含义
Sno	CHAR(6)	学号
Cno	CHAR(4)	课号
Grade	INT	成绩

{Sno, Cno}是SC的主键, Sno是SC的外键, 参照Student关系的主键Sno

```
CREATE TABLE SC (  
Sno CHAR(6),  
Cno CHAR(4),  
Grade INT,  
PRIMARY KEY (Sno, Cno),  
FOREIGN KEY (Sno) REFERENCES Student(Sno));
```

声明用户定义完整性约束

- 功能:
 - ① 规定属性值非空: **NOT NULL**
 - ② 规定属性值不重复: **UNIQUE**
 - ③ 规定数值型属性值自动递增: **AUTO_INCREMENT**
 - ④ 定义属性的缺省值: **DEFAULT** 缺省值
 - ⑤ 规定属性值必须满足表达式给出的条件: **CHECK (表达式)**²

Example (声明用户定义完整性约束)

```
CREATE TABLE Student (  
Sno CHAR(6),  
Sname VARCHAR(10) NOT NULL,  
Ssex ENUM('M', 'F') NOT NULL CHECK (Ssex IN ('M', 'F')),  
Sage INT DEFAULT 0 CHECK (Sage >= 0),  
Sdept VARCHAR(20),  
PRIMARY KEY (Sno)  
);
```

²MySQL只解析CHECK, 但存储引擎并不处理

删除关系

- 功能: 删除关系
- SQL语句: **DROP TABLE** 关系名1, 关系名2, ..., 关系名n;
- 删除关系时会连同关系中的数据一起删除!

修改关系模式

- 功能: 修改关系模式的定义
 - ① 增加、修改、删除属性
 - ② 增加、删除约束
- SQL语句: **ALTER TABLE**

Example (增加属性)

在Student关系中增加属性Mno, 记录学生的班长的学号

```
ALTER TABLE Student (ADD Mno CHAR(6));
```

Example (增加约束)

将属性Mno声明为Student关系的外键, 参照Student的主键Sno

```
ALTER TABLE Student (  
  ADD CONSTRAINT fk_mno  
  FOREIGN KEY (Mno) REFERENCES Student(Sno)  
);
```

修改关系模式(续)

Example (删除属性)

删除Student关系的Mno属性

```
ALTER TABLE Student (DROP Mno);
```

Example (删除约束)

删除Student关系中Mno属性上的外键约束fk_mno

```
ALTER TABLE Student (DROP CONSTRAINT fk_mno);
```

Example (修改属性定义)

将Student关系中Sname属性的类型修改为VARCHAR(20)且不重名^a

```
ALTER TABLE Student (ALTER Sname VARCHAR(20) UNIQUE);
```

^aMySQL使用ALTER TABLE Student (MODIFY Sname VARCHAR(20) UNIQUE);

定义视图(View)

讲完SQL查询后再回来讲 ▶ SQL数据查询

- 基本关系: 关系数据库中真实存储的关系
- 视图: 从用户的视角所看到的数据
- 功能: 创建、修改、删除视图
- SQL语句: CREATE VIEW、ALTER VIEW、DROP VIEW

创建视图

语法: CREATE VIEW 视图名 [(属性名列表)] AS 子查询;^a

- 例: 为选修了“Database Systems”课的计算机系(CS)的学生建立视图, 列出学号、姓名和成绩

```
CREATE VIEW CS_Student_on_DB AS
SELECT Sno, Sname, Grade
FROM Student NATURAL JOIN SC
WHERE Sdept = 'CS' AND Cname = 'Database Systems';
```

^aMySQL对CREATE VIEW中的子查询有很多限制条件, 参考<https://dev.mysql.com/doc/refman/5.5/en/create-view.html>

定义视图(续)

修改视图定义

语法: **ALTER VIEW** 视图名 [(属性名列表)] **AS** 子查询;

- 例: 修改视图CS_Student_on_DB, 增加性别属性

```
ALTER VIEW CS_Student_on_DB AS
SELECT Sno, Sname, Ssex, Grade
FROM Student NATURAL JOIN SC
WHERE Sdept = 'CS' AND Cname = 'Database Systems';
```

删除视图

语法: **DROP VIEW** 视图名;

视图查询

语法: 与SQL查询的语法相同

- 例: 查询计算机系(CS)没有通过“Database Systems”课考试的学生

```
SELECT Sno, Sname FROM CS_Student_on_DB WHERE Grade < 60;
```

定义视图(续)

学习视图的概念有助于具体理解第1章讲过的“三层模式结构”

- 关系模式就是概念模式(conceptual schema), 视图就是外模式(external schema)
- 一个关系数据库只有一个概念模式, 但可以有多多个外模式
- **CREATE VIEW**定义了从外模式(视图)到概念模式的映射
- 若概念模式发生变化, 只需用**ALTER VIEW**重新定义外模式(视图)到概念模式的映射, 而无需修改基于视图编写的查询, 从而实现了逻辑数据独立性(logical data independence)

3.2 SQL数据更新

SQL Data Modification

SQL数据更新

- 插入数据
 - ▶ 直接插入元组
 - ▶ 插入子查询结果(讲完SQL查询后再讲)
- 修改数据
 - ▶ 基于本关系的修改
 - ▶ 基于其他关系的修改(讲完SQL查询后再讲)
- 删除数据
 - ▶ 基于本关系的删除
 - ▶ 基于其他关系的删除(讲完SQL查询后再讲)
- 数据完整性检查
 - ▶ 实体完整性检查
 - ▶ 参照完整性检查
 - ▶ 用户定义完整性检查
- 更新视图★

直接插入元组

- 功能: 向一个关系中直接插入元组
- 语法: **INSERT INTO 关系名 [(属性名1, ..., 属性名n)] VALUES (表达式1, ..., 表达式n);**
- 说明:
 - ▶ 列出的属性名与表达式个数相同
 - ▶ 表达式*i*的值是该元组属性*i*的值
 - ▶ 若没给出属性名列表, 则表示插入一条完整的元组, 其属性顺序与关系模式中定义的属性顺序相同
 - ▶ 若只列出部分属性名, 则该元组的其他属性值为空

Example (插入元组)

- ① **INSERT INTO Student
VALUES ('232102', 'Cincy', 'F', 18, 'Math');**
- ② **INSERT INTO Student (Sno, Sname, Sage, Ssex)
VALUES ('232102', 'Cincy', 18, 'F');**

基于本关系的数据修改

- 功能: 修改一个关系中的数据
- 语法: **UPDATE 关系名 SET 属性名1 = 表达式1, ..., 属性名n = 表达式n [WHERE 修改条件];**
- 说明:
 - ▶ 若WHERE子句存在, 则将满足修改条件的元组中属性*i*的值修改为表达式*i*的值
 - ▶ 若WHERE子句不存在, 则将所有元组中属性*i*的值修改为表达式*i*的值
 - ▶ 修改条件仅涉及该关系本身

Example (修改数据)

- ① 将学号为232102的学生的所在院系修改为数学系(Math)
UPDATE Student SET Sdept = 'Math' WHERE Sno = '232102';
- ② 将所有学生的年龄增加1岁
UPDATE Student SET Sage = Sage + 1;

基于本关系的数据删除

- 功能: 删除一个关系中的元组
- 语法: **DELETE FROM 关系名 [WHERE 删除条件];**
- 说明:
 - ▶ 若WHERE子句存在, 则将满足删除条件的元组从关系中删除
 - ▶ 若WHERE子句不存在, 则将所有元组从关系中删除(这与DROP TABLE不同, 因为它并不删除关系本身)
 - ▶ 删除条件仅涉及该关系本身

Example (修改数据)

- ① 将学号为232102的学生元组删除

```
DELETE FROM Student WHERE Sno = '232102';
```

数据完整性检查

- 目的: 数据修改可能导致数据库违反完整性约束, 需要对数据完整性进行检查
- 功能: 检查数据库是否满足完整性约束
 - ① 实体完整性检查
 - ② 参照完整性检查
 - ③ 用户定义完整性检查

实体完整性检查

- 插入元组或对主键中属性的值进行修改时检查
- 检查主键中各属性的值是否为空。只要有一个为空, 就拒绝插入或修改
- 检查主键值是否唯一。若不唯一, 则拒绝插入或修改

用户定义完整性检查

- 插入或修改元组时进行检查
- 若不满足用户定义完整性约束, 则拒绝插入或修改该元组

参照完整性检查

有四种情况可能会破坏参照完整性

情况1

在参照关系(如SC)中插入一个元组, 该元组的外键(如Sno)的值在被参照关系(如Student)的主键(如Sno)值中找不到, 即悬空(dangling)。若发生这种情况, 则拒绝插入该元组。

SC		
Sno	Cno	Grade
221101	1002	92
221101	1003	85
221101	1006	88
231101	1006	90
231102	1003	80
232103	1002	100

拒绝插入

Student				
Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math
232102	Cincy	F	18	Math

参照完整性检查(续)

情况2

修改参照关系(如SC)中的一个元组, 造成该元组的外键(如Sno)的值在被参照关系(如Student)的主键(如Sno)值中找不到。若发生这种情况, 则拒绝修改该元组。

SC		
Sno	Cno	Grade
221101	1002	92
221101	1003	85
221101	1006	88
231101	1006	90
231102	1003	80
232101	1002	100
232103		

拒绝修改

Student				
Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math
232102	Cincy	F	18	Math

参照完整性检查(续)

情况3

从被参照关系(如Student)中删除一个元组，造成参照关系(如SC)中某些元组的外键(如Sno)的值在被参照关系(如Student)的主键(如Sno)值中找不到。如果发生这种情况，则采取下列处理方法之一：

- ① 当参照关系(如SC)中没有任何元组的外键值与被参照关系(如Student)中待删除元组的主键值相对应时，方可删除改元组；否则拒绝删除该元组。
- ② 级联(cascade)删除参照关系(如SC)中相关联的元组
- ③ 将参照关系(如SC)中相关联元组的外键(如Sno)值置为空(NULL)

SC		
Sno	Cno	Grade
221101	1002	92
221101	1003	85
221101	1006	88
231101	1006	90
231102	1003	80
232101	1002	100

Student				
Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math
232102	Cincy	F	18	Math

拒绝删除

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

参照完整性检查(续)

情况4

修改被参照关系(如Student)中一个元组，造成参照关系(如SC)中某些元组的外键(如Sno)的值在被参照关系(如Student)的主键(如Sno)值中找不到。如果发生这种情况，则采取下列处理方法之一：

- ① 当参照关系(如SC)中没有任何元组的外键值与被参照关系(如Student)中待修改元组的主键值相对应时，方可修改改元组；否则拒绝修改该元组。
- ② 级联(cascade)修改参照关系(如SC)中相关联的元组
- ③ 将参照关系(如SC)中相关联元组的外键(如Sno)值置为空(NULL)

SC		
Sno	Cno	Grade
221101	1002	92
221101	1003	85
221101	1006	88
231101	1006	90
231102	1003	80
232101	1002	100

Student				
Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math
232102	Cincy	F	18	Math

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

参照完整性检查(续)

- 在FOREIGN KEY子句中声明，当违反参照完整性约束时，如何处理
- 语法: 在FOREIGN KEY子句的末尾加上

ON DELETE RESTRICT | CASCADE | SET NULL | NO ACTION

ON UPDATE RESTRICT | CASCADE | SET NULL | NO ACTION

- ▶ RESTRICT或NO ACTION: 拒绝删除或修改，缺省处理方式
- ▶ CASCADE: 级联删除或修改
- ▶ SET NULL: 置为空值

Example (参照完整性检查)

- ① FOREIGN KEY (Sno) REFERENCES Student(Sno)
ON DELETE RESTRICT
ON UPDATE RESTRICT

插入查询结果

讲完SQL查询后再回来讲 [▶ SQL数据查询](#)

- 功能: 将一个查询的全部结果插入到另一个关系中
- 语法: INSERT INTO 关系名 (子查询);

Example (修改数据)

- ① 将计算机系学生的学号、姓名、选课数、平均分插入到关系CS_Grade(Sno, Sname, Amt, AvgGrade)中
INSERT INTO CS_Grade
(SELECT Sno, Sname, COUNT(*), AVG(Grade)
FROM Student NATURAL JOIN SC
GROUP BY Sno, Sname);

基于外部关系的数据修改

- 待修改的元组在一个关系中，而修改条件涉及其他的关系

Example (修改数据)

将计算机系(CS)全体学生的成绩置零

- ① UPDATE SC SET Grade = 0 WHERE 'CS' =
(SELECT Sdept FROM Student WHERE Student.Sno = SC.Sno);
- ② UPDATE SC SET Grade = 0 WHERE Sno IN
(SELECT Sno FROM Student WHERE Sdept = 'CS');
- ③ UPDATE SC SET Grade = 0 WHERE EXISTS
(SELECT * FROM Student
WHERE Student.Sno = SC.Sno AND Sdept = 'CS');

基于外部关系的数据删除

- 待删除的元组在一个关系中，而删除条件涉及其他的关系

Example (修改数据)

删除计算机系所有学生的选课记录

- ① DELETE FROM SC WHERE 'CS' =
(SELECT Sdept FROM Student WHERE Student.Sno = SC.Sno);
- ② DELETE FROM SC WHERE Sno IN
(SELECT Sno FROM Student WHERE Sdept = 'CS');
- ③ DELETE FROM SC WHERE EXISTS
(SELECT * FROM Student
WHERE Student.Sno = SC.Sno AND Sdept = 'CS');

SQL数据查询

- ① 单关系查询
- ② 连接查询
- ③ 嵌套查询

3.3.1 单关系查询

投影查询

- 功能: 从一个关系中选出指定的列
- 语法: **SELECT [DISTINCT] 属性名列表 FROM 关系名;**
- 说明:
 - ▶ 查询结果中属性的顺序与属性名列表给出的属性顺序相同
 - ▶ 不加**DISTINCT**, 则不去除结果中的重复元组; 加上**DISTINCT**, 则去除结果中的重复元组
 - ▶ 若要返回关系中所有的列, 可将属性名列表简写为*

Example (投影查询)

- ① 查询学生的学号和姓名

```
SELECT Sno, Sname FROM Student;
```
- ② 查询所有系名

```
SELECT DISTINCT Sdept FROM Student;
```
- ③ 查询全部学生信息

```
SELECT * FROM Student;
```

投影查询(续)

- 可将投影查询中的属性名替换为表达式, 做更复杂的投影操作
- 语法: **SELECT [DISTINCT] 表达式列表 FROM 关系名;**
- 说明:
 - ▶ 表达式可以是常量、算术表达式、函数表达式、逻辑表达式
 - ▶ 查询结果中表达式列的名称就是该表达式的字符串, 可以用“**表达式 AS 属性名**”将表达式列重命名

Example (扩展的投影查询)

- ① 查询学生的学号和姓名(姓名全大写)

```
SELECT Sno, UPPER(Sname) FROM Student;
```
- ② 查询学生的姓名和出生年份

```
SELECT Sname, (2019 - Sage) AS bd FROM Student;
```

Example (在线练习)

- ① Find the model numbers and the hard disk size (in GB) of all PC's

选择查询

- 功能: 从一个关系中选择满足给定条件的元组
- 语法: **SELECT** [**DISTINCT**] 表达式列表 **FROM** 关系名 **WHERE** 选择条件;

Example (选择查询)

- ① 查询计算机系(CS)全体学生的学号和姓名

```
SELECT Sno, Sname FROM Student WHERE Sdept = 'CS';
```
- ② 查询计算机系(CS)全体男同学的学号和姓名

```
SELECT Sno, Sname FROM Student  
WHERE Sdept = 'CS' AND Ssex = 'M';
```

Example (在线练习)

- ① What PC models have a speed of at least 3.00?
- ② What PC models have a speed of at least 3.00 and ram of at least 1024MB?

选择查询条件

表达式比较

- 语法: 表达式1 比较运算符 表达式2
- 比较运算符: =, <, <=, >, >=, !=, <> (不等于)

范围比较

- 语法: 表达式1 [**NOT**] **BETWEEN** 表达式2 **AND** 表达式3
- 功能: 判断表达式1的值是否(不)在表达式2和表达式3之间

集合元素判断

- 语法: 表达式1 [**NOT**] **IN** (表达式2, ..., 表达式n)
- 功能: 判断表达式1的值是否(不)在表达式2, ..., 表达式n的值构成的集合中

选择查询条件(续)

字符串匹配

- 语法: 字符串表达式 [NOT] LIKE 模式 [ESCAPE 转义字符]
- 功能: 判断字符串表达式的值是否匹配给定的含有通配符的模式
 - ▶ 通配符_: 匹配单个字符
 - ▶ 通配符%: 匹配任意长度的字符串(含空串)
 - ▶ 可通过ESCAPE子句指定转义字符, 默认转义字符为\

Example (字符串匹配)

- ① 查询姓名首字母为E的学生的学号和姓名

```
SELECT Sno, Sname FROM Student WHERE Sname LIKE 'E%';
```

- ② 查询姓名为4个字母且首字母为E的学生的学号和姓名

```
SELECT Sno, Sname FROM Student WHERE Sname LIKE 'E___';
```

选择查询条件(续)

字符串正则表达式匹配

- 语法: 字符串表达式 [NOT] REGEXP|RLIKE 模式
 - ▶ MySQL使用关键词REGEXP或RLIKE
 - ▶ Oracle使用关键词REGEXP_LIKE
 - ▶ MS SQL Server使用关键词LIKE
- 功能: 判断字符串表达式的值是否匹配给定的正则表达式
- 正则表达式
 - ▶ MySQL正则表达式
<https://dev.mysql.com/doc/refman/5.5/en/regexp.html>

Example (正则表达式)

- ① 查询姓名首字母为E或F的学生的学号和姓名

```
SELECT Sno, Sname FROM Student  
WHERE Sname REGEXP '^[EF].*';
```

选择查询条件(续)

空值(NULL)判断

- 语法: 属性名 IS [NOT] NULL
- 功能: 判断属性值是否(不)为空
- 错误写法: 属性名 = NULL, 属性名 <> NULL, 属性名 != NULL。
这些错误写法的结果均为UNKNOWN (既非真, 也非假)

Example (空值判断)

查询选了课但还未取得成绩的学生

```
SELECT Sno FROM SC WHERE Grade IS NULL;
```

选择查询条件(续)

逻辑运算

- 逻辑运算符: AND, OR, NOT
- 逻辑运算真值表

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN

- 只有使查询条件为真的元组才能出现在查询结果中

集合操作

- 功能: 求两个查询语句结果的并、交、差
- 语法:
 - ① 求并: 查询语句1 UNION [ALL] 查询语句2
 - ② 求交: 查询语句1 INTERSECT 查询语句2⁴
 - ③ 求差: 查询语句1 MINUS/EXCEPT 查询语句2⁵
- 说明:
 - ▶ 查询语句1的结果的属性名将作为集合操作结果的属性名
 - ▶ 若使用关键词ALL, 则并集不去重
 - ▶ 即使两个查询语句的结果都是有序的, 集合操作的结果也未必有序

Question

*** 在MySQL中如何实现INTERSECT和EXCEPT?

⁴Oracle和MS SQL Server支持INTERSECT, MySQL不支持INTERSECT

⁵Oracle用MINUS, MS SQL Server用EXCEPT, MySQL两者都不支持

查询结果排序

- 功能: 按照指定的顺序, 对查询结果关系中的元组进行排序
- 语法: 在查询语句的后面加上ORDER BY 属性名1 [ASC|DESC], ..., 属性名n [ASC|DESC]
 - ▶ 按照(属性1, ..., 属性n)的字典序进行排序
 - ▶ ASC表示升序(默认), DESC表示降序
- 若排序属性含空值, ASC: 属性值为空的元组排在最后; DESC: 属性值为空的元组排在最前
- 通常用ORDER BY子句对最终查询结果排序, 而不对中间结果排序

Example (结果排序)

- ① 查询计算机系(CS)全体学生的学号和姓名, 并按学号升序排列

```
SELECT Sno, Sname FROM Student WHERE Sdept = 'CS'  
ORDER BY Sno;
```

- ② 查询全体学生的信息, 结果按所在系升序排列, 同一个系的学生按年龄降序排列

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

限制查询结果数量

- **功能**: 限制查询结果中元组的数量(不是标准SQL的内容)
- **语法**: 在查询语句的后面加上**LIMIT [偏移量,] 结果数量**或者**LIMIT 结果数量 [OFFSET 偏移量]**⁶
- 从偏移量(默认是0)位置的元组开始, 返回指定数量的元组

Example (限制查询结果数量)

- ① 查询1001号课得分最高的前10名学生的学号

```
SELECT Sno FROM SC WHERE Cno = '1001'  
ORDER BY Grade DESC LIMIT 10;
```

⁶Oracle和MS SQL Server使用不同的语法

聚集(Aggregation)查询

- **功能**: 计算一个关系上某表达式所有值的聚集值(值的个数COUNT、最大值MAX、最小值MIN、总和SUM、平均值AVG)⁷
- **语法**: **SELECT 聚集函数([DISTINCT] 表达式) FROM...WHERE...**

COUNT(*)	所有元组的数量
COUNT(表达式)	非空表达式值的数量
COUNT(DISTINCT 表达式)	不同的非空表达式值的数量
MAX([DISTINCT] 表达式)	表达式的最大值
MIN([DISTINCT] 表达式)	表达式的最小值
SUM(表达式)	表达式值的和
SUM(DISTINCT 表达式)	不同的表达式值的和
AVG(表达式)	表达式值的平均值
AVG(DISTINCT 表达式)	不同的表达式值的平均值

⁷MySQL还支持其他聚集函数, 如GROUP_CONCAT(拼接)、VARIANCE(方差)、STD(标准差) <https://dev.mysql.com/doc/refman/5.5/en/group-by-functions.html>

聚集(Aggregation)查询(续)

Example (聚集查询)

- ① 查询计算机系全体学生的数量

```
SELECT COUNT(*) FROM Student WHERE Sdept = 'CS';
```

- ② 查询计算机系学生的最大年龄

```
SELECT MAX(Sage) FROM Student WHERE Sdept = 'CS';
```

- 聚集函数不能出现在WHERE子句中! (重要的事情说三遍)
- 例: 查询年龄最大的学生的学号

```
SELECT Sno FROM Student WHERE Sage = MAX(Sage);
```

写法错误, 正确的做法是使用嵌套查询(第3.3.3节讲)

分组(Group By)查询

- 功能: 关系代数的分组操作
- 语法: **SELECT 分组属性列表, 聚集函数表达式列表 FROM 关系名 WHERE 选择条件 GROUP BY 分组属性列表;**
- 说明:
 - ① 根据指定的分组属性, 对一个关系中的元组进行分组, 分组属性值相同元组的分为一组
 - ② 对每个组中元组的非分组属性的值进行聚集
 - ③ 分组查询语句中不能包含分组属性及聚集函数表达式以外的其他表达式(为什么?)

Example (分组查询)

- ① 统计每门课的选课人数和平均成绩

```
SELECT Cno, COUNT(*), AVG(Grade) FROM SC GROUP BY Cno;
```

- ② 统计每个系的男生人数和女生人数

```
SELECT Sdept, Ssex, COUNT(*) FROM Student  
GROUP BY Sdept, Ssex;
```

分组(Group By)查询(续)

- 分组完成后，经常需要按照组内元组的统计信息对分组进行筛选
- 语法: SELECT 分组属性列表, 聚集函数表达式列表 FROM 关系名 WHERE 选择条件 GROUP BY 分组属性列表 HAVING 分组筛选条件;

Example (分组查询)

- ① 查询选修了3门以上课程的学生的学号和选课数

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

- ② 查询3门以上课程得分超过90的学生的学号及这些课程的平均分

```
SELECT Sno, AVG(Grade) FROM SC WHERE Grade >= 90  
GROUP BY Sno HAVING COUNT(*) >= 3;
```

分组(Group By)查询(续)

注意事项

- SELECT子句的目标列中只能包含分组属性和聚集函数⁸
- WHERE子句的查询条件中不能出现聚集函数
- HAVING子句的分组筛选条件中可以使用聚集函数
- WHERE、GROUP BY和HAVING的执行顺序
 - ① 按照WHERE子句给出的条件，从关系中选出满足条件的元组
 - ② 按照GROUP BY子句指定的分组属性，对元组进行分组
 - ③ 按照HAVING子句给出的条件，对分组进行筛选

Example (在线练习)

- ① How many models does every manufacturer have?
- ② Find those hard-disk sizes that occur in two or more PC's

⁸单纯出于性能考虑，MySQL允许某些非分组属性出现在目标列

3.3.2 连接查询

内连接

- 功能: 按照给定的连接条件, 对两个关系做内连接
- 语法: `SELECT ... FROM 关系名1 [INNER] JOIN 关系名2 ON 连接条件`

Example (内连接)

- ① 查询学生及其选课情况, 列出学号、姓名、课号、得分

```
SELECT Student.Sno, Sname, Cno, Grade
FROM Student JOIN SC ON (Student.Sno = SC.Sno);
```
- ② 查询选修了“Database Systems”的学生的学号、姓名和分数

```
SELECT Student.Sno, Sname, Grade
FROM Student JOIN SC ON (Student.Sno = SC.Sno)
JOIN Course ON (SC.Cno = Course.Cno)
WHERE Cname = 'Database Systems';
```

内连接(续)

- 当内连接是等值连接，且连接属性同名时，可使用如下语法
- 语法: 关系名1 [INNER] JOIN 关系名2 USING (连接属性列表)

Example (内连接)

- ① 查询学生及其选课情况，列出学号、姓名、课号、得分

```
SELECT Student.Sno, Sname, Cno, Grade  
FROM Student JOIN SC USING (Sno);
```

- ② 查询选修了“Database Systems”的学生的学号、姓名和分数

```
SELECT Student.Sno, Sname, Grade  
FROM Student JOIN SC USING (Sno) JOIN Course USING (Cno)  
WHERE Cname = 'Database Systems';
```

Example (在线练习)

- ① What PC models with a price less than \$500 does the manufacturer A produce?
- ② What manufactures make laptops with a hard disk of at least 100GB?

自然连接

- 功能: 两个关系做自然连接
- 语法: 关系名1 NATURAL JOIN 关系名2

Example (自然连接)

- ① 查询学生及其选课情况，列出学号、姓名、课号、得分

```
SELECT Student.Sno, Sname, Cno, Grade  
FROM Student NATURAL JOIN SC;
```

- ② 查询选修了“Database Systems”的学生的学号、姓名和分数

```
SELECT Student.Sno, Sname, Grade  
FROM Student NATURAL JOIN SC NATURAL JOIN Course  
WHERE Cname = 'Database Systems';
```

Example (在线练习)

- ① ★★ Find the manufacturers of PC's with at least three different speeds

自连接

- 功能: 一个关系与其自身进行连接
- 语法: 与其他连接操作相同
- 注意:
 - ▶ 参与连接的关系在物理上是同一个关系, 但在逻辑上看作两个关系, 因此用AS必须重命名
 - ▶ 当为关系取了别名后, 但凡用到该关系时都必须使用其别名, 不能再使用原关系名
 - ▶ 属性名前必须加别名做前缀

Example (自连接)

- ① 查询和Elsa在同一个系学习的学生的学号和姓名(含Elsa)

```
SELECT S2.Sno, S2.Sname
FROM Student AS S1 JOIN Student AS S2
ON S1.Sdept = S2.Sdept
WHERE S1.Sname = 'Elsa';
```

自连接(续)

Example (在线练习)

- ① Find those pairs of PC models that have both the same speed and RAM (a pair should be listed only once)
- ② Find the model numbers of all printers that are cheaper than the printer model 3002 (用连接)
- ③ Find those hard-disk sizes that occur in two or more PC's (用连接)

外连接

- 功能: 两个关系做外连接
- 语法:
 - ① 左外连接: 关系名1 LEFT [OUTER] JOIN 关系名2 ON 连接条件
 - ② 右外连接: 关系名1 RIGHT [OUTER] JOIN 关系名2 ON 连接条件
 - ③ 全外连接: 关系名1 FULL [OUTER] JOIN 关系名2 ON 连接条件⁹
 - ④ 自然外连接: 关系名1 NATURAL LEFT|RIGHT [OUTER] JOIN 关系名2
- 当外连接是等值连接, 且连接属性同名时, 可使用 USING (连接属性列表) 声明连接条件

Example (内连接)

- ① 查询没有选课的学生的学号和姓名

```
SELECT Student.Sno, Sname
FROM Student LEFT JOIN SC ON (Student.Sno = SC.Sno)
WHERE Cno IS NULL;
```

⁹Oracle和MS SQL Server支持FULL OUTER JOIN, MySQL不支持FULL OUTER JOIN 🔍

外连接(续)

Question

★如何在MySQL中实现FULL OUTER JOIN?

Example (练习)

- ① ★★ Find the PC model with the highest available speed

3.3.3 嵌套查询

嵌套查询

- **查询块**: 一个SELECT-FROM-WHERE语句称为一个查询块(block)
- **嵌套查询**: 将一个查询块嵌套在另一个查询块中得到的查询称为嵌套查询(nested query), 内层查询块称为子查询(subquery)
- **子查询的类型**
 - ▶ **不相关子查询(independent subquery)**: 子查询不依赖于外层查询
 - ▶ **相关子查询(correlated subquery)**: 子查询依赖于外层查询

Example (子查询的类型)

查询和Elsa在同一个系学习的学生的学号和姓名(含Elsa)。似曾相识?

① 使用不相关子查询

```
SELECT Sno, Sname FROM Student WHERE Sdept =  
(SELECT Sdept FROM Student WHERE Sname = 'Elsa');
```

② 使用相关子查询

```
SELECT Sno, Sname FROM Student AS S WHERE EXISTS  
(SELECT * FROM Student AS T  
WHERE T.Sname = 'Elsa' AND T.Sdept = S.Sdept);
```

嵌套查询的写法

- ① 在集合判断条件中使用子查询: 使用 **[NOT] IN**
- ② 在比较条件中使用子查询: 使用 **比较运算符**
- ③ 在存在性测试条件中使用子查询: 使用 **[NOT] EXISTS**
- ④ 子查询结果作为派生关系

Example (子查询的写法)

查询和Elsa在同一个系学习的学生的学号和姓名(含Elsa)

- ① 在集合判断条件中使用子查询

```
SELECT Sno, Sname FROM Student WHERE Sdept IN  
(SELECT Sdept FROM Student WHERE Sname = 'Elsa');
```

- ② 在比较条件中使用子查询

```
SELECT Sno, Sname FROM Student WHERE Sdept =  
(SELECT Sdept FROM Student WHERE Sname = 'Elsa');
```

- ③ 在存在性测试条件中使用子查询

```
SELECT Sno, Sname FROM Student AS S WHERE EXISTS  
(SELECT * FROM Student AS T  
WHERE T.Sname = 'Elsa' AND T.Sdept = S.Sdept);
```

嵌套查询写法1: 在集合判断条件中使用子查询

- **功能**: 判断表达式的值是否(不)属于子查询的结果
- **语法**: **表达式 [NOT] IN (子查询)**
- 先执行子查询, 后执行父查询

Example (嵌套查询的写法)

- ① 查询和Elsa在同一个系学习的学生的学号和姓名(含Elsa)

```
SELECT Sno, Sname FROM Student WHERE Sdept IN  
(SELECT Sdept FROM Student WHERE Sname = 'Elsa');
```

- ② 查询选修了“Database Systems”的学生的学号和姓名

```
SELECT Sno, Sname FROM Student WHERE Sno IN  
(SELECT Sno FROM SC WHERE Cno IN  
(SELECT Cno FROM Course  
WHERE Cname = 'Database Systems'));
```

很显然, 这个嵌套查询既不易读, 也不高效。有更好的做法吗?

嵌套查询写法2: 在比较条件中使用子查询

- **功能:** 将表达式的值与子查询的结果进行比较
- **语法:** **表达式 比较运算符 [ALL|ANY|SOME] (子查询)**
 - ▶ **ALL:** 当表达式的值与子查询结果中任意的值都满足比较条件时, 返回真; 否则, 返回假
 - ▶ **ANY或SOME:** 当表达式的值与子查询结果中某个值满足比较条件时, 返回真; 否则, 返回假
 - ▶ 如果子查询结果应仅包含单个值, 则无需在比较运算符后面加ALL、ANY或SOME
- 先执行子查询, 后执行父查询

Example (嵌套查询的写法)

- ① 查询和Elsa在同一个系学习的学生的学号和姓名(含Elsa)

```
SELECT Sno, Sname FROM Student WHERE Sdept =  
(SELECT Sdept FROM Student WHERE Sname = 'Elsa');
```

嵌套查询写法2: 在比较条件中使用子查询(续)

Example (嵌套查询的写法)

- ① 查询年龄最大的学生的学号(第2次出现, 上次是什么时候出现的?)

```
SELECT Sno FROM Student WHERE Sage =  
(SELECT MAX(Sage) FROM Student);
```

- ② 查询比计算机系(CS)全体学生年龄都大的学生的学号

```
SELECT Sno FROM Student WHERE Sno > ALL  
(SELECT Sage FROM Student WHERE Sdept = 'CS');
```

- ③ 查询学生平均年龄比全校学生平均年龄大的系

```
SELECT Sdept FROM Student GROUP BY Sdept  
HAVING AVG(Sage) > (SELECT AVG(Sage) FROM Student);
```

Example (练习)

- ① Find the model numbers of all printers that are cheaper than the printer model 3002 (第2次出现, 上一次是怎么做的? 哪个做法更好?)

嵌套查询写法3: 在存在性测试条件中使用子查询

- 功能: 判断子查询结果是否(不)为空
- 语法: **[NOT] EXISTS (子查询)**
- 子查询的SELECT后无需列出目标列, 只需用SELECT *, 因为我们只判断子查询结果是否为空, 并不需要使用子查询结果

Example (嵌套查询的写法)

- ① 查询和Elsa在同一个系学习的学生的学号和姓名(含Elsa)

```
SELECT Sno, Sname FROM Student AS S
WHERE EXISTS (SELECT * FROM Student AS T
WHERE T.Sname = 'Elsa' AND T.Sdept = S.Sdept);
```

查询执行过程

- ① 从Student关系中依次取出每条元组 t , 设 t 的Sdept属性值为 $t[Sdept]$
- ② 将 $t[Sdept]$ 代入子查询后, 执行子查询
- ③ 若子查询的结果不为空, 则将 t 投影到Sno和Sname属性上, 并输出投影后的元组

用EXISTS实现全称量词 \forall 功能

- SQL不支持全称量词 \forall (for all)
- 可以用EXISTS实现全称量词, 因为 $\forall x(P(x)) = \neg \exists x(\neg P(x))$

Example (全称量词)

- ① 查询选修了全部课程的学生的学号(思考一下用关系代数怎么做?)

```
SELECT Sno FROM Student WHERE NOT EXISTS (
SELECT * FROM Course WHERE NOT EXISTS (
SELECT * FROM SC
WHERE SC.Sno = Student.Sno AND SC.Cno = Course.Cno));
```

思路: 设 $t \in Student$ 是某个选修了全部课程的学生的元组,

则 $\forall c \in Course$, 必 $\exists s \in SC$, 使 $s[Sno] = t[Sno] \wedge s[Cno] = c[Cno]$
这等价于:

$\neg \exists c \in Course$, 使得 $\neg \exists s \in SC$, 使 $s[Sno] = t[Sno] \wedge s[Cno] = c[Cno]$

用EXISTS实现逻辑蕴含 \rightarrow 功能

- SQL不支持逻辑蕴含 \rightarrow (implication)
- 可以用EXISTS实现逻辑蕴含，因为 $x \rightarrow y = \neg x \vee y$

Example (全称量词)

- ① 查询至少选修了231101号学生选修的全部课程的学生们的学号

```
SELECT Sno FROM Student WHERE NOT EXISTS (  
  SELECT * FROM SC AS SC1  
  WHERE SC1.Sno = '231101' AND NOT EXISTS (  
    SELECT * FROM SC AS SC2  
    WHERE SC1.Sno = Student.Sno AND SC2.Cno = SC1.Cno));
```

嵌套查询写法4: 子查询结果作为派生关系

- 派生表(derived table): 将子查询的结果当作关系放在外层查询的FROM子句中使用，称为派生表
- 语法: **FROM (子查询)**
- 子查询必须是独立子查询
- 派生表必须重命名

Example (嵌套查询的写法)

- ① 查询选修了3门以上课程的学生们的学号和选课数

```
SELECT Sno, Amt  
FROM (SELECT Sno, COUNT(*) AS Amt FROM SC GROUP BY Sno)  
WHERE Amt >= 3;
```

(第2次出现，上一次是怎么做的？还能怎么做?)

SQL视图定义和数据更新

讲完SQL查询了，该回头把SQL视图定义和数据更新讲完了

- SQL视图定义 ▶ SQL视图定义
- SQL数据更新 ▶ SQL数据更新

总结

① SQL数据定义

- ▶ 基本数据类型: 数值型、日期时间型、字符串型、枚举型
- ▶ 创建关系模式: CREATE TABLE
- ▶ 修改关系模式: ALTER TABLE
- ▶ 删除关系模式: DROP TABLE
- ▶ 定义视图: CREATE VIEW, ALTER VIEW, DROP VIEW

② SQL数据更新

- ▶ 插入数据: INSERT
- ▶ 修改数据: UPDATE
- ▶ 删除数据: DELETE

③ SQL数据查询

- ▶ 单关系查询: SELECT ... FROM ... WHERE
- ▶ 集合查询: UNION, INTERSECT, MINUS/EXCEPT
- ▶ 连接查询: INNER JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN
- ▶ 嵌套查询: IN, 比较运算符, EXISTS, 派生表

习题

- ① 身份证号用CHAR(13)和VARCHAR(13)哪种类型表示更合适?
- ② 什么情况下需要使用UNION ALL? 什么情况下使用UNION?
- ③ MySQL不支持集合交操作INTERSECT, 如何实现集合交操作?
- ④ MySQL不支持集合差操作EXCEPT, 如何实现集合差操作?
- ⑤ 为什么分组查询语句的目标属性只能包含分组属性和聚集函数?
- ⑥ 相关子查询和不相关子查询有何区别?
- ⑦ 为什么子查询中通常不使用DISTINCT和ORDER BY?
- ⑧ 视图和基本表有什么区别?

资源

- MySQL Community Server下载:
<https://dev.mysql.com/downloads/mysql/>
- MySQL文档: <https://dev.mysql.com/doc/>
- 在线SQL练习:
<https://dbis-uibk.github.io/relax/calc.htm>¹⁰
- W3Schools SQL教程: <https://www.w3schools.com/sql/>

¹⁰该系统功能尚不完善, 请在MySQL上练习

致谢

- 感谢孔美豪同学发现课件中的问题
- 感谢龚利锋、王梓宣同学提供课堂练习题笔记