第3章 关系数据库标准语言SQL

3.1 SQL概述

了解

3.1.1 SQL的发展历程

(1) 1970年：IBM研究中心E. F. Codd连续发表多篇论文，提出了关系数据模型。

(2) 1972年：IBM开始研制实验型关系数据库管理系统System R，配套的查询语言为SQUARE (Specifying Queries As Relation Expression)，数学符号较多。

(3) 1974年：Boyce和Chamberlin把SQUARE语言修改为SEQUEL(Structured English Query Language)，去掉了一些数学符号，并采用英语单词表示和结构式的语法规则。

(4) 20世纪70年代末：由于使用方便、功能丰富、语言简洁易学，SQL语言得到了很快的应用和推广。

SEQUEL简称为SQL(Structured Query Language)，称为结构式查询语言，具有定义、查询、更新和控制等多种功能。

(5) 1986年：美国国家标准局颁布了美国标准的SQL语言；1987年被国际标准化组织采纳为国际标准，这两个标准称为SQL 86。

(6) 1989年：ISO颁布了增强完整性特征的SQL 89标准。在SQL 86和SQL 89标准中，基本表没有关键字的概念，并采用索引机制弥补。

(7) 1992年：ISO对SQL 89进行了大量的修改和扩充，推出了SQL 92。

(8) 1997年以后：成为动态网站(Dynamic web content)的后台支持。

(9) SQL/99：核心级别跟其他8种相应的级别，包括递归查询，程序跟流程控制，基本的对象(object)支持包括oids。

(10) SQL/2003：包含了XML相关内容，自动生成列值(column values)。

(11) 2005年：Tim O'eilly提出了Web 2.0理念，称数据将是核心，SQL将成为“新的HTML”。

(12) SQL/2006：定义了SQL与XML(包含XQuery)的关联应用。

(13) 2006年：Sun公司以SQL基础的数据库管理系统嵌入Java V6。

理解 3.1.2 SQL 数据库的体系结构

三级结构	外模式	概念模式	内模式		
关系模型术语	子模式	关系模式	存储模式	元组	属性
SQL 术语	视图	基本表	存储文件	行	列

SQL 数据库由表构成，表可以是基本表，也可以是视图。在用户看来，视图和基本表是一样的，都是关系（表格）。**基本表**是实际存储在数据库中的表。

一个基本表可以跨一个或多个存储文件，一个存储文件也可以在存放一个或多个基本表，无须一一对应。

在 SQL 中，外模式一级**数据结构**的基本单位是视图，视图是从若干个基本表和（或）其他视图构造出来的，所以视图又被称为**虚表**。

理解 3.1.3 SQL 的组成

（1）数据定义：SQL DDL，用于定义 SQL 模式、基本表、视图和索引。

（2）数据操纵：SQL DML，又分为数据查询和数据更新两大类。

（3）数据控制：该部分包括对基本表和视图的授权、完整性规则的描述、事务控制等。

（4）嵌入式 SQL：该部分内容涉及 SQL 语句嵌入在宿主语言程序中使用的规则。

3.2 SQL 的数据定义

应用 3.2.1 SQL 数据定义语句

操作对象	操作方式		
	创　建	修　改	删　除
数据库	CREATE DATABASE	ALTER DATABASE	DROP DATABASE
表	CREATE TABLE	ALTER TABLE	DROP TABLE
视图	CREATE VIEW	ALTER VIEW	DROP VIEW
索引	CREATE INDEX	ALTER INDEX	DROP INDEX

3.2.2 SQL 的基本数据类型

- 二进制数据类型：binary、varbinary、image
- 数字数据类型
  - 精确数字
    - 精确整数：bigint、int、smallint、tinyint
    - 精确小数：decimal、numeric
  - 近似数字：float、real
- 字符数据类型：char、varchar、text
- Unicode数据类型：nchar、nvarchar、ntext
- 日期和时间数据类型：datetime、smalldatetime
- 货币数据类型：money、smallmoney
- 其他数据类型：bit、timestamp、uniqueidentifier 等

详见附录B。

3.2.3 基本表的创建、修改和撤销

(1) 基本表的创建：

CREATE TABLE 基本表名

(列名,数据类型…完整性约束)

完整性约束有三种子句：

- 主键子句：PRIMARY KEY。
- 检查子句：CHECK。
- 外键子句：FOREIGN KEY。

SQL 允许列的值为空值,但要求主键不能为空。如果要求列不能为空值,则应在数据类型后加“NOT NULL”说明。基本表创建好后,用 INSERT 命令把数据插入基本表中。

(2) 基本表结构的修改：

- ALTER TABLE 基本表名 ADD 列名 数据类型：在表中增加一列。
- ALTER TABLE 基本表名 DROP 列名：从表中删除一列。
- ALTER TABLE 基本表名 ALTER COLUMN 列名：在表中修改一列。

(3) 基本表的删除：

DROP TABLE 基本表名。

一个基本表删除后,其所有的数据也一并丢失。

3.2.4 视图的创建和撤销

(1) 视图的创建：

```
CREATE VIEW 视图名(列名序列) 列名序列可省略
AS SELECT 查询语句
```

(2) 视图的撤销：

```
DROP VIEW 视图名
```

如用户经常要用到 S、SC、C 的其中几列，可以单独创建一个视图：

```
CREATE VIEW SS
(sno,sname,cno,cname,grade )
AS SELECT S.sno,sname,C.cno,cname ,grade
FROM S, SC,C
WHERE S. sno=SC.sno AND SC.cno=C.cno
```

3.2.5 索引的创建和撤销

在 SQL 86 和 SQL 89 中，基本表中没有关键字的概念，用索引机制弥补，索引属于存储路径的概念；在 SQL 中使用主键的，在基本表中定义主键。

(1) 索引的创建：

```
CREATE[UNIQUE]INDEX 索引名 ON 基本表名(列名)
```

UNIQUE：要求列的值在基本表中不重复。

一个索引键也可以对应多个列。缺省为升序(ASC)排列，也可用降序(DESC)排列。

(2) 索引的撤销：

```
DROP INDEX 索引名
```

3.3 SQL 的数据查询

数据查询是关系运算理论在 SQL 语言中的主要体现，本节从 SELECT 语句的基本句法、完整句法和各种限定三方面进行说明。

应用 3.3.1 SELECT 语句的基本句法

SELECT A1, …,An
FROM R1 ,…Rm
WHERE F

- 算术比较运算符：<，<=，=，!=，>，>=，<>
- 逻辑运算符：AND，OR，NOT
- 集合运算符：UNION(并)，INTERSECT(交)，EXCEPT(差)
- 集合成员运算符：IN，NOT IN
- 谓语：EXISTS(存在量词)，ALL，SOME，UNIQUE(唯一)
- 聚合函数：AVG，MIN，MAX，SUM，COUNT，
  聚合函数不允许复合操作

(1) 其中，F 为条件表达式，比关系代数中的公式更灵活。

(2) SELECT 语句可以嵌套使用，层次分明，具有结构程序设计特点。

(3) 嵌套查询比连接查询的笛卡儿积效率高。

例如：查询选修了课程号为“C1”的学生的学号及姓名。

联接查询： SELECT S. sno,sname FROM S, SC WHERE S. sno=SC. sno AND Cno='c1'	嵌套查询 1： SELECT sno,sname FROM S WHERE sno IN (SELECT sno FROM SC WHERE cno='c1')	嵌套查询 2： SELECT sno, sname FROM S WHERE c1 IN (SELECT cno FROM SC WHERE sno=S. sno)	嵌套查询 3： SELECT sno, sname FROM S WHERE EXISTS (SELECT * FROM SC WHERE sno=S. sno AND cno='c1')

3.3.2 SELECT 语句的完整句法

SELECT 目标表的列名或列表达式序列
FROM 基本表和/或视图序列
[WHERE 行条件表达式] 行条件子句
[GROUP BY 列名序列] 分组子句
[HAVING 组条件表达式] 组条件子句
[ORDER BY 列名 ASC|DESC] 排序子句

- []是指该成分可有可无；
- 读取 FROM 子句中基本表、视图的数据，执行笛卡儿积操作；
- 选取满足 WHERE 子句中给出的条件表达式的元组；
- 按 GROUP BY 子句中指定列分组，同时提取满足 HAVING 子句中条件表达式的那些组；
- 按 SELECT 子句中给出的列名或列表达式求值输出；
- ORDER BY 子句对输出的目标表进行排序，ASC(升序排列)或 DESC(降序排列)。

分组子句： 例：按照不同年龄统计选修课程的人数。去掉重复数据 SELECT age,COUNT(DISTINCT SC.sno) FROM S, SC WHERE S. sno=SC. sno GROUP BY age (把满足 WHERE 子句中条件的查询结果按年龄分组；此时的 SELECT 语句应对每一组分开进行操作)	排序子句： 例：输出女生在 50 人以上不同年龄的人数分布情况，并按人数升序、年龄降序排列。 SELECT age, COUNT (sno) FROM S WHERE sex='女' GROUP BY age HAVING COUNT (*)>50 (组条件子句，去掉小于等于 50 人的组) ORDER BY 2, age DESC (对 SELECT 子句中的第 2 列按升序排列，如人数相同的有多条数据，则多条数据再按年龄降序排列。)

3.3.3 SELECT 语句中的各种限定

(1) DISTINCT：如果要求输出的表格中不许出现重复元组，那么可在 SELECT 后加一保留字 DISTINCT。

(2) SELECT 子句允许包含＋、－、＊、/，以及列名、常数的算术表达式。

(3) 列和基本表的改名操作：如一个基本表在 SELECT 语句中多次出现，则可以用"SELECT 旧名 AS 新名"来改名，以方便操作。

(4) 字符串的匹配操作：匹配操作符是"LIKE "，表达式中可以有两个通配符％(表示 0 个、或个字符)和_(表示 1 个字符)。

例如，检索以 D 开头的学生姓名：

```
WHERE Sname LIKE 'D%'
```

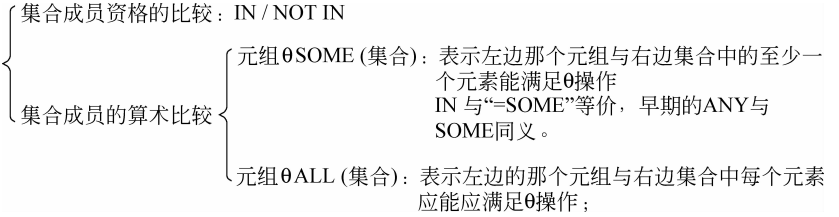
(5) 如两个子查询结果的结构完全一致时，可让两个子查询执行并、交、差操作。

例如，

```
(SELECT 查询语句 1  )
UNION  [ALL]-带 ALL 时，不消除重复元组
(SELECT 查询语句 2  )
```

(6) 空值的比较操作：SQL 中允许列值为空，空值用保留字 NULL 表示。涉及空值的比较操作，返回值是 false。

(7) 集合的比较操作



(8) 空关系的测试：EXISTS (集合)/NOT EXISTS (集合)，测试一个集合是否为空/非空，当集合非空时，返回逻辑值 true/false。

(9) 重复元组的测试：UNION (集合)/NOT UNION(集合)，测试集合中是否有重复元组，如有重复元组则返回 false/true。

3.4 SQL 的数据更新

应用

3.4.1 INSERT 插入语句

(1) 元组值的插入：

```
INSERT INTO 基本表名 (列名序列)
VALUES (元组值)
```

例如：

```
INSERT INTO SC (sno,cno,grade )
VALUES ('s1','c1',90)
```

(2) 查询结果的插入：

```
INSERT INTO 基本表名(列名序列)
SELECT 查询语句
```

应用 3.4.2 DELETE 删除语句

SQL 的删除操作指从基本表中删除元组。

```
DELETE  FROM 基本表名
[WHERE 条件表达式]
```

先执行 WHERE 子句中的子查询，然后再对查找到的元组执行删除操作。

```
DELETE  FROM  SC
WHERE cno IN
(SELECT cno
FROM C
WHERE cname='DB')
```

应用 3.4.3 UPDATE 修改语句

需要修改基本表中的某些列值时：

```
UPDATE 基本表名
SET 列名 1=值表达式 1[,列名 2=值表达式 2…]
[WHERE 条件表达式]
```

把 C1 课程的任课老师的姓名改为张明：

```
UPDATE  C
SET  teacher='张明'
WHERE cno='c1'
```

把女同学的成绩提高 10%：

```
UPDATE SC
SET grade=grade * 1.1
WHERE sno IN
(SELECT sno FROM S WHERE sex='女')
```

理解 3.4.4 对视图的更新操作规则

对于视图的查询操作同基本表，没有什么限制；但对于视图中的元组的更新操作(INSERT/DELETE/UPDATE)有如下限定规则：

(1) 如果视图是从单个基本表使用选择或投影操作导出的，且包含了基本表的主键或候选键，那么这个视图称为行列子集视图。该视图允许更新操作。

(2) 如果一个视图是从多个基本表使用连接操作导出的，那么该视图不允许执行更新操作。

(3) 如果在导出视图的过程中，使用了分组和聚合操作，则该视图也不允许执行更新操作。

(4) 在 SQL 2 中，允许更新的视图在定义时，必须加上 WITH CHECK OPTION 短语。

3.5 嵌入式 SQL

理解 3.5.1 SQL 运行环境

(1) 交互式 SQL：在终端交互方式下使用；不可直接使用指针、数组等操作；

(2) 嵌入式 SQL：嵌入在高级语言的程序中使用；不能直接进行集合的操作。

实现嵌入式 SQL，有两种处理方式：

- 扩充宿主语言的编译程序，使之能处理 SQL 语句；
- 采用预处理方式，先用预处理程序对源程序进行扫描，识别出 SQL 语句，并处理成宿主语言的函数调用形式。通常 DBMS 提供一个 SQL 的函数定义库，供编译时调用。

SQL 与宿主语言的接口：是通过事先由宿主语言程序定义的共享变量实现的，SQL STATE 是特殊的共享变量。

理解 3.5.2 嵌入式 SQL 的使用

(1) 在程序中要区分 SQL 语句与宿主语言语句。

嵌入式 SQL 语句的格式为：

```
EXEC SQL<SQL 语句>END_EXEC
```

(2) 允许嵌入的 SQL 语句引用宿主语言的程序变量(共享变量)，但引用这些变量前，应加“：”以示区别；这些变量在宿主语言的程序中定义，并用 DECLARE 说明：

```
EXEC SQL BEGIN DECLARE SECTION: char sno [5],name[9];
    char SQLSTATE [6];
EXEC SQL END DECLARE SECTION:
```

(3) SQL 的集合处理方式与宿主语言单记录处理方式间的协调：SQL 语句处理的是记录集合，而宿主语言一次只处理一个记录。

因此引入游标(cursor：指针)机制，把集合操作转换成单记录处理方式。

游标是与某一查询结果相联系的符号名，相关的 SQL 语句如下。

(1) 游标定义语句：

```
DECLARE  EXEC SQL DECLARE<游标名>
CURSOR FOR<SELECT 语句>END _EXEC
```

定义中的 SELECT 语句并不立即执行。

(2) 游标打开语句：

```
OPEN  EXEC SQL OPEN<游标名>END_EXEC
```

执行游标定义中的 SELECT 语句。

(3) 游标推进语句：

```
FETCH EXEC SQL FETCH FROM<游标名>INTO<变量表>END_EXEC
```

将游标推进一行，并把游标指向的当前行的值取出，送共享变量。

(4) 游标关闭语句：

```
CLOSE EXEC SQL CLOSE<游标名>END_EXEC
```

3.5.3 嵌入式 SQL 的使用技术

对于 SQL DDL 语句，只要加上前缀标识 EXEC SQL 和结束标识 END_EXEC，就能嵌入在宿主语言中使用。

对于 SQL DML 语句，要注意是否使用了游标机制。

1. 不涉及游标的 SQL DML 语句

(1) INSERT、DELETE 和 UPDATE 语句，同 DDL 只需加上 EXEC SQL 前缀和 END_SQL 即可。

```
EXEC SQL INSERT INTO S (sno,sname ,age )
VALUES (: givensno ,: sn ,: sa );
```

(2) 对于查询结果是单元组的 SELECT 语句，应加 INTO 子句，用以指出找到的值应送到相应的共享变量中去。

```
EXEC SQL SELECT sname ,age ,sex INTO : sn,: sa ,: ss
    FROM S
```

```
WHERE sno=:givensno ;
```

sn ,sa ,ss, givensno 都是共享变量,已在主程序中定义,并用 SQL 的 DECLARE 语句说明过。

2. 涉及游标的 SQL DML 语句

当 SELECT 语句查询的结果是多个元组时,此时宿主语言无法使用,必须使用游标机制,把多个元组一个一个地传送给宿主语言程序处理,具体过程如下。

(1) 先用游标定义语句,定义一个游标与某个 SELECT 语句对应。

(2) 游标用 OPEN 语句打开后,处于活动状态,此时,游标指向查询结果的第一个元组之前。

(3) 每执行一次 FETCH 语句,游标指向下一个元组,并把值送到共享变量,供程序处理。

(4) 最后用 CLOSE 语句关闭游标。

在游标处于活动状态下,可以修改或删除游标指向的元组。用 WHERE CURRENT OF <游标名>来表示游标指向的当前元组。

3. 卷游标的定义和推进

游标在查询时只能从头到尾一行一行推进,使用不便,故给出了卷游标的概念。

定义句法如下:

```
EXEC SQL DECLARE<游标名>SCROLL CURSOR FOR<SELECT 语句>
END _EXEC
```

卷游标的推进句法:

EXEC SWL FETCH { NEXT / PRIOR / FIRST / LAST / RELATIVE <n> / ABSOLUTE <n> } FROM<游标名>INTO<变量表>END_EXEC

相对当前游标n行的元组;

查询结果是绝对第n行的元组;

第4章 关系数据理论

4.1 关系模式的设计问题

理解 4.1.1 关系模式的外延和内涵

关系模式是用来定义关系的，一个关系数据库包含一组关系，定义这些关系的关系模式的全体构成该数据库的模式。

(1) 关系模式的外延：通常所说的关系、(数据库)实例或当前值。由于元组的插入、删除和修改，它随着时间的推移在不断变化。

(2) 关系模式的内涵：与时间独立，包括关系、属性以及域的一些定义和说明，还有各种数据完整性约束(静态约束和动态约束)。

- 静态约束：包括各种数据之间的联系(称为**数据依赖**)、主键的设计和关系值的各种限制，用于定义关系的有效数据问题。
- 动态约束：主要定义如插入、删除和修改等各种操作的影响。

关系数据库是以关系模型为基础的数据库，它利用关系来描述现实世界。关系数据库设计理论主要包含三个方面的内容：

(1) 数据依赖。

(2) 范式。

(3) 模式设计方法。

4.2 函数依赖 FD

理解 4.2.1 FD 的定义

设有关系模式 R(U)，X、Y 是 U 的子集，r 是 R 的任一具体关系，如果对 r 的任意两个元组 t_1，t_2，由 $t_1[X]=t_2[X]$，会导致 $t_1[Y]=t_2[Y]$，则称 **X 函数决定 Y** 或 **Y 函数依赖于 X**，记为 $X \rightarrow Y$。FD 是对关系 R 的一切可能的当前值 r 定义的，不是针对某个特定关

系。如 Sno→Sname，指的是每个学号都只能确定一个学生姓名。在当前值 r 的两个不同元组中，如果 X 值相同，就一定要求 Y 值也相同，Y 值由 X 值决定，这种数据依赖称为**函数依赖 FD**。

掌握

4.2.2 函数依赖的类型

(1) 非平凡的函数依赖：如果 X→Y，但 Y 不是 X 的子集，则称 X→Y 是非平凡的函数依赖。

反之为平凡的函数依赖，平凡的函数依赖不反映新的语义。

(2) 部分函数依赖：若 X→Y，且存在 X 的真子集 X_1，使 X_1→Y，则称 Y 部分依赖于 X。记作 $X \xrightarrow{P} Y$。

(3) 完全函数依赖：若 X→Y，且对于 X 的任何一个真子集 X_1，都不存在 X_1→Y，则称 Y 完全依赖于 X。记作 $X \xrightarrow{f} Y$。

(4) 传递函数依赖：若 X→Y，且 Y→Z，而 Y→X，则 X→Z，则称 Z 对 X 传递函数依赖。

理解

4.2.3 FD 的逻辑蕴涵，FD 集的闭包 F^+

如有关系模式 R(X,Y,Z)，它的函数依赖集 F={X→Y,Y→Z}。如果 A→B，而 B→C，问 A→C 是否成立？这就是函数依赖的逻辑蕴涵问题。

设 F 是关系模式 R 的一个函数依赖集，X、Y 是 R 的属性子集，如果从 F 中的函数依赖能够推出 X→Y，则称 **F 逻辑蕴涵** X→Y，记为 F|=X→Y。被 F 逻辑蕴涵的函数依赖的全体构成的集合，称为 F 的闭包，记为 F^+，F^+={X→Y |F|=X→Y}。

理解

4.2.4 键和 FD 的联系

键是唯一地标识实体的属性集。

设有关系模式 R(A_1,A_2,…,A_n)，F 是 R 上的函数依赖集，X 是{A_1,A_2,…,A_n}的一个子集，如果：

(1) X→$A_1A_2\cdots A_n \in F^+$；表示 X 能唯一决定一个元组；

(2) 且不存在 X 的真子集 Y，使得 Y→$A_1A_2\cdots A_n$ 成立，则称 X 是 R 的一个候选键。表示 X 满足条件 1 且无多余的属性集。

理解

4.2.5 FD 的推理规则

阿氏公理：函数依赖有一个正确和完备的推理规则，1974 年，W. W. Armstrong 总结了各种推理规则，把其中最主要、最基本的作为公理，称为 Armstrong 推理规则系统。

设有关系模式 $R(A_1,A_2,\cdots,A_n)$，F 是 R 上的函数依赖集，属性集 $U=\{A_1,A_2,\cdots,A_n\}$，X、Y、Z、W 是 U 的子集：

(1) 基理：

- 规则 1 自反律：如果 $Y\subseteq X\subseteq U$，由 $X\rightarrow Y$ 为 F 所蕴含；
- 规则 2 增广律：如果 $X\rightarrow Y$ 为 F 所蕴涵，且 $Z\subseteq U$，由 $XZ\rightarrow YZ$ 为 F 所蕴含；
- 规则 3 传递律：如果 $X\rightarrow Y$ 和 $Y\rightarrow Z$ 为 F 所蕴含，则 $X\rightarrow Z$ 为 F 所蕴含。

(2) 推理：

- 规则 4 伪传递律：如果 $X\rightarrow Y$ 和 $WY\rightarrow Z$ 在 R 上成立，则 $WX\rightarrow WY\rightarrow Z$ 在 R 上成立；
- 规则 5 合并律：如果 $X\rightarrow Y$ 和 $X\rightarrow Z$ 在 R 上成立，则 $X\rightarrow YZ$ 在 R 上成立；
- 规则 6 分解律：如果 $X\rightarrow Y$ 和 $Z\subseteq Y$ 成立，则 $X\rightarrow Z$ 在 R 上成立。

理解 4.2.6 FD 推理规则的完备性

函数依赖推理规则系统——自反律、增广律和传递律是完备的，即不能从 F 中使用推理规则导出的函数依赖不在 F^+ 中。

理解 4.2.7 属性集闭包的计算

F 是属性集合 U 上的一个函数依赖集，设 $X\in U$，则称所有用阿氏公理推出的函数依赖 $X\rightarrow A$ 中所有 A 的集合，称为**属性集 X 关于 F 的闭包，记为 X^+**。

计算函数依赖集合 F 的闭包 F^+，实际上是不可能的，判断 $X\rightarrow Y$ 是否在 F^+ 中，只要判断 $X\rightarrow Y$ 能否用推理规则从 F 导出，即**判断 $Y\in X^+$**。

理解 4.2.8 FD 集的等价和覆盖

关系模式 R(U)上的两个函数依赖集 F 和 G，如果满足 $F^+=G^+$，则称 F 和 G 是等价的，亦称 F 覆盖 G 或 G 覆盖 F。

- 每个函数依赖 F 都可以被一个右部只有单属性的函数依赖集 G 所覆盖。
- 每个函数依赖 F 都有最小覆盖(最小函数依赖集合 Fmin，右边都是单属性，且 F 中的任一函数依赖 $X\rightarrow A$，其 $F^-(X\rightarrow A)$ 与 F 不等价)。

4.3 关系模式的分解特性

理解 4.3.1 模式分解中存在的问题

在分解处理中会涉及一些新问题，如原模式所满足的特性在新模式中是否被保持。为保持原有模式所满足的特性(如属性间的约束)，要求分解处理具有**无损连接性**和**保持**

函数依赖性。否则无法保证关系数据的完整性。

理解

4.3.2 无损连接

设 R 为一关系模式，分解成关系模式 $\rho=\{R1,R2,\cdots,Rk\}$，F 是 R 上的一个函数依赖集。如果对 R 中满足 F 的每一个关系 r 都有：$r=\pi_{R1}(r)\bowtie\pi_{R2}(r)\bowtie\cdots\bowtie\pi_{Rk}(r)$，即 r 为它在 Ri 上的投影的**自然连接**，则称这个分解 ρ 相对于 F 是**无损连接分解**。

其中 $\pi_{Ri}(r)$表示关系 r 在模式 Ri 的属性上的投影。**无损连接**即分解后的关系自然连接后完全等于分解前的关系，则这个分解相对于 F 是无损联接分解。

掌握

4.3.3 无损连接的测试方法

设有关系模式 $R=(A_1,A_2,\cdots,A_n)$，F 是 R 上的函数依赖集，R 的一个分解 $\rho=\{R1,R2,\cdots,Rk\}$，判断其是否无损连接，方法如下：

(1) 构造一张 k 行 n 列的表格，每列对应一个属性 A_j，每行对应一个模式 Ri。如果 A_j在 Ri 中，那么在表格的每 i 行第 j 列填上符号 a_j，否则填上符号 b_{ij}。

(2) 反复检查 F 的每一个函数依赖，并修改表格中的元素，如下：

取 F 中的函数依赖 X→Y，如果表格中有两行在 X 分量上相等，在 Y 分量上不相等，那么修改 Y，使这两行在 Y 分量上也相等。

如果 Y 的分量中，有一个是 a_j，那么另一个也修改到 a_j；否则用 b_{ij}替换另一个符号(把下标 ij 改成较小的数)。

重复以上的过程，直至整个表格不能再修改为止。

(3) 若修改到最后一张表格中，有一行是全 a，即 $a_1a_2\cdots a_n$，那么，ρ 相对于 F 是无损连接分解。

如果 R1∩R2→R1-R2/R2-R1 在函数依赖集 F 中，则 R1、R2 的分解是无损连接。

理解

4.3.4 保持 FD 的分解

保持 FD 的分解，可以保证关系能由它的那些投影进行自然连接而恢复。如果关系模式在分解后不能保持函数依赖，那么在数据库中就会出现异常现象。

保持关系模式的一个分解是等价的，另一个重要条件是关系模式的函数依赖集在分解后仍在数据库模式中保持不变，即关系模式 R 到 $\rho=\{R1,R2,\cdots,Rk\}$的分解应使函数依赖集 F 被 F 在这些 Ri 上的投影蕴涵。

由于 F 中的依赖是对关系模式 R 的完整性约束，因此要求 R 分解后也要保持 F，称为**分解 ρ 保持函数集 F**。否则，我们用 $\rho=\{R1,R2,\cdots,Rk\}$表达 R 时，可能会找到一个数据库实例能满足投影后的依赖，但不满足 F，而导致关系数据的错误。

4.4 关系模式的范式

理解

4.4.1 范式的定义

满足特定要求的模式称为范式。

(1) 1NF：如果关系模式 R 的所有属性的值域中的每一个值都是不可再分解的值，则称 R 属于**第一范式(1NF)**。即**要求其属性值不可再分成更小的部分**。

不能满足上述条件的关系称为非规范化关系。非 1NF 的关系模式的缺点是更新操作困难。

(2) 2NF：如果关系模式 R 为第一范式，且 R 中每一个非主属性完全函数依赖于 R 的某个候选键，则称 R 属于**第二范式(2NF)**。

如果非主属性对键的函数依赖不完全，则执行数据库操作时会出现插入异常、删除异常、更新异常以及数据冗余等问题。

(3) 3NF：如果关系模式 R 是第二范式，且**每个非主属性**都不**传递依赖**于 R 的候选健，则称 R 属于**第三范式(3NF)**。

(4) BCNF：如果关系模式 R 是第一范式，且**每个属性**(包括主属性和非主属性)都不传递依赖于 R 的候选键，则称 R 为 BC **范式**。BCNF 是第三范式的改进形式。

BCNF 在数据冗余、插入、修改和删除中具有较好的特性。如果模式 R 是 BCNF，则模式 R 必定是第三范式，反之，则不一定成立。

理解

4.4.2 分解成 BCNF 模式集的算法

对于任一关系模式，都可找到一个分解使之达到 3NF，且具有无损连接和保持函数依赖性。而对模式进行 BCNF 分解可以保证无损连接，但不一定能保持函数依赖集。

设有关系模式 R，F 是 R 的函数依赖集，要求输出 R 的无损连接 ρ{R1，R2，…，Rk}；分解成 BCNF 的方法如下：

(1) 置初值 ρ={R}；

(2) 如果 ρ 中所有关系模式都是 BCNF，则转到(4)；

(3) 如果 ρ 中有一个关系模式 S 不是 BCNF，则 S 中必能找到一个函数依赖 X→A，有 X 不是 S 的**键**，且 A∈X；

(4) 分解结束，输出 ρ。

掌握

4.4.3 分解成 3NF 模式集的算法

设有关系模式 R，F 是 R 的最小函数依赖集，要求输出 R 的一个分解 ρ{R1，R2，…，Rk}；分解成 3NF 的方法如下：

(1) 如果 R 中的某些属性在 F 的所有依赖的左边和右边都不出现，那么这些属性可以从 R 中分出去，单独构成一个关系模式；

(2) 如果 F 中有一个依赖 X→A，有 XA=R，则 ρ={R}，转到(4)；

(3) 对于 F 中每一个 X→A，构成一个关系模式 XA。如果 F 中有 X→A1，X→A2，…，X→An，则可以用模式 X A1A2 …An 代替 n 个模式 XA1，XA2，…，XAn；

(4) 分解结束，输出 ρ。

例如：关系模式 W(C,T,H,R,S,G)满足 F=|C->T,CS->G,HR->C,HS->R |，分解为 3NF 如下：ρ={CT,CSG,HRC,HSR}。

掌握

4.4.4 模式设计方法的原则

一个好的模式设计方法应符合下列三条原则。

(1) 表达性：分解的关键问题是要等价地分解，涉及到两个数据库模式的等价性问题(数据等价和依赖等价)，分别用无损联接和保持函数依赖性来衡量；

(2) 分离性：指属性间的独立联系应该用不同的关系模式表达。独立的联系独立表示；如模式中存在 X→Y，我们就把(XY)作为一个基本信息单位；

(3) 最小冗余性：要求在分解后的数据库能表达原来数据库的所有信息这个前提下实现，目的是节省存储空间，提高对关系的操作效率，清除不必要的冗余。

理解

4.4.5 多值依赖及 4NF

函数依赖能表达属性值间的多对一联系，但不能表示属性值间的一对多联系(如一个学校有多个学生)，如要刻画一对多联系，则需要使用**多值依赖**概念。

把间接联系的属性放在一个模式中，就会出现我们所不希望产生的问题，如数据冗余和操作异常。

设 R 是一个关系模式，D 是 R 上的多值依赖集合。如果 D 中成立非平凡多值依赖 X→→Y 时，X 必是 R 的超键，那么称 R 是**第四范式(4NF)**。

一个模式如果属于 4NF，则必定属于 BCNF。

掌握

4.4.6 关系模式规范化过程

关系模式的规范化过程如图 4.1 所示。

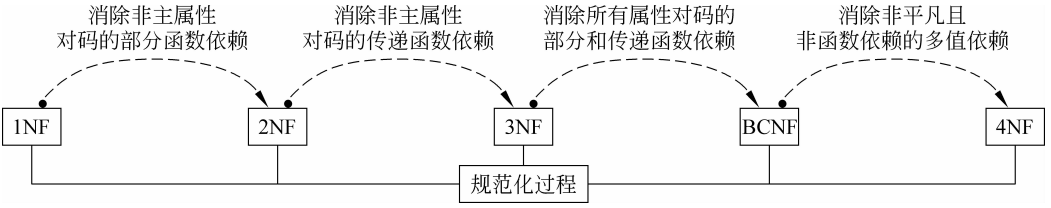


图 4.1 关系模式规范化过程