数据管理基础

> FXNB!

概论

- DataBase Management System：数据库管理系统
- DataBase：数据库
- DataBase User：数据库用户
  - 终端用户
    - 临时用户
    - 缺乏经验的用户
  - 应用程序员
  - 数据库管理员
- Data Model：数据模型
  - Hierarchical：层次数据模型（树）
  - Network：网状数据模型（图）
  - Relational：关系模型
  - Object-Oriented：面向对象模型
  - Object-Relational：对象关系模型

关系模型

术语

- SQL （关系模型） [文件系统]
- 表 （关系） [记录文件]
- 列名 （属性） [域]
- 行 （元组） [记录]
- 表头 （模式） [记录类型]

数据独立性

查询语句与表内容无关，仅与列名有关

域

Domain(A) = { 列名A的取值范围 }

列类型

问题：

- 大部分商业数据库不支持枚举类型
- char(13) 和 char(14) 无法进行比较

关系规则

1. 第一范式规则：关系模型不允许含有多值属性（某行某列有多个值）和含有内部结构的列（列类型是一个对象）
2. 只能基于内容存取行（行列是没有顺序的）原子性
3. 行唯一性规则：关系中的任何两个元组在同一时刻不能是完全相同的
4. 实体完整性：主键列的取值不允许为空值

键和超键

- 超键：一组可以唯一确定行的列
- 键：最小的超键，没有任何一个真子集是超键
- 表的主键：可以唯一标识行
    - 定理 2.4.2：每个表至少有一个键

Null

- 空值：当一个具体值未知或不合适时填充表
    - 不能够进行比较（ **等于** 也不行！）

相容表

Compatible Table

表头相同的表称为相容表 Head(R) = Head(S)

- 列类型不能不同
- 列数量不能不同
- 列名不能不同
- 列顺序 **可以** 不同

只有相容表才可以做 交、并、差 的操作

关系操作

- 投影（projection）[π]
    - 取一列（重复只算一次）
    - $\pi_{name}(table)$
- 选择（selection）
    - table WHERE age > 18
- 连接（join）[∞]
    - Head(T1) = {A1...An, B1...Bn}
    - Head(T2) = {B1...Bn, C1...Cn}
    - Head(T1∞T2) = {A1...An, B1...Bn, C1...Cn}
    - 只包含 T1(Bi) = T2(Bi) 的行
    - 定理
        - 若 $Head(T_1) \bigcap Head(T_2) = \phi$，则 $T_1 \infty T_2 = T_1 \times T_2$
        - 若 $Head(T_1) = Head(T_2)$，则 $T_1 \infty T_2 = T_1 \bigcap T_2$
        - 若 $Head(T_1 \subseteq Head(T_2)$，则 $T_1 \infty T_2 \subseteq T_2$
- division
    - Head(T1) = {A1...An, B1...Bn}
    - Head(T2) = {B1...Bn}

- Head(T1÷T2) = {A1...An}
- 只包含 T1(Bi) = T2(Bi) 且将 Bi 列去除的行
- 定理
  - 若 $R = T \times S$，则 $T = R \div S; S = R \div T$
  - 若 $T = R \div S$，则 $T \times S \subseteq R$

关系完备

SQL 语言如果能达成和关系操作同样的功能，就称其为 **关系完备** 的

外连接

- 连接：无法反向重构原来的表
- 外连接：左外连接 + 右外链接（右 / 左表缺失值为 null）
  - 可以反向重构
- Theta连接
  - $R\infty_F S = (R \times S)whereF$

ORSQL

定义新的数据类型

```
CREATE TYPE name_t AS OBJECT(
    Iname varchar(30)
    fname varchar(30)
)
/*用已定义的数据类型来创建新类型*/
CREATE TYPE person_t AS OBJECT(
    ssno   int,
    pname  name_t
)
```

用对象数据类型创建表

```
CREATE TABLE people OF person_t(
    PRIMARY KEY(ssno)
)
```

对象的赋值与查询

```
INSERT INTO people VALUES (101, name_t('Hao', 'XingWei'))
```

```
SELECT p.pname.Iname FROM people p WHERE p.ssno = 101
```

```
UPDATE people p SET p.pname = name_t('Ding', 'YaoXin') WHERE p.ssno = 101
UPDATE people p SET p.pname.Iname = 'Hao' WHERE p.ssno = 101
UPDATE people p SET p = person_t(
                            111,
                            name_t('Fei', 'ZhengQian')
                        )
WHERE ssno = 101
```

对象的引用

```
CREATE TYPE order_t AS OBJECT(
    ordno    int
    ordcust  REF customer_t
)
/*创建含有引用类型的关系表*/
/*此前已建立好 customers 表*/
CREATE TABLE orders OF order_t(
    PRIMARY KEY(ordno)
    SCOPE FOR(ordcust) IS customers
)
```

```
SELECT o.ordno FROM orders o WHERE o.ordcust.cid = 1
```

```
/*REF 返回对象的引用指针*/
SELECT c.cname FROM customers c WHERE NOT EXISTS(
    SELECT * FROM orders o WHERE o.ordcust = REF(c)
)
```

```
/*嵌套定义*/
CREATE TYPE police_officer_t AS OBJECT(
    pol_person   person_t
    badge_number int
    partner REF police_officer_t
)
/*DEREF 检索整个被引用对象，而非仅仅获得引用指针*/
SELECT VALUES(p), DEREF(p.partner) FROM police_officers p
```

```
/*ISDANGLING 用于判断引用的元组对象是否存在*/
/*若不存在返回 TRUE*/
SELECT o.cid FROM orders o WHERE o.ordcust IS DANGLING
SELECT o.cid FROM orders o WHERE o.ordcust <> (SELECT REF(c) FROM customers c
WHERE c.cid = o.cid)
```

有关 REF 的约束

- 在删除类型之前要先删除表
- 删除类型：DROP TYPE people_t FORCE

```
UPDATE orders o SET ordcust = (SELECT REF(c) FROM customers c WHERE c.cid =
o.cid)
/* VALUE() 可以获取一个对象的值 */
INSERT INTO police_officers
    SELECT VALUE(p), 101, REF(p0)
    FROM people p, police_officers p0
    WHERE p.ssno = 1 AND p0.badge_number = 66
```

TABLE TYPE

```
CREATE TYPE dependents_t AS TABLE OF person_t
/*嵌套表*/
CREATE TABLE employees(
    eid  int,
    dependents dependents_t
) NESTED TABLE dependents STORE AS dependents_t
/*访问嵌套表*/
/*一定要使用转换函数 table() */
SELECT eid FROM employees e WHERE 6 < (SELECT COUNT(*) FROM table(e.dependents))
/*Oracle 没有提供 nested table 的比较运算! */
```

```
/*Oracle 提供了对象类型的比较*/
SELECT eid FROM employees e WHERE name_t('Wei', 'YuXi') IN (SELECT d.pname FROM
table(e.dependents) d)
/*Oracle 不支持嵌套表统计查询*/
SELECT COUNT(e.dependents) FROM employees e WHERE eid = 1 /*是错误的! */
/*可以使用 CURSOR 实现*/
SELECT eid, CURSOR(SELECT COUNT(*) FROM table(e.dependents)) FROM employees e
/*这样也可以，那还要 CURSOR 干嘛？？？*/
SELECT eid, SELECT COUNT(*) FROM table(e.dependents) FROM employees e
```

ARRAY TYPE

```
CREATE TYPE extensions_t AS VARRAY(4) OF int
```

可以使用 table() 将一个 VARRAY 属性转化成嵌套表

- VARRAY 有序，Nested Table 无序
- 成员最大数目确定；成员数目没有限制
- 直接存储在表中；有单独的存储表

成员函数

```
CREATE FUNCTION sum_n(n int) return int IS
    i int;
    total int := 0;
    begin
        for i in 1...n Loop
            total := total + i;
        end loop;
        return total;
    end;
CREATE TYPE rectangle_t AS OBJECT(
```

```
    pt1 point_t,
    pt2 point_t,
    member function inside(p point_t)
        return int
)
```

ESQL

过程

- 声明
- 条件处理
- SQL 连接数据库
- 主程序
- SQL 断开连接

声明

- 编译时检查类型
- 提前分配内存空间

```
EXEC SQL BEGIN DECLARE SECTION;
    char cust_id[5];
    float cust_discnt;
EXEC SQL END DECLARE SECTION;
```

条件处理

```
EXEC SQL WHENEVER SQLERROR GOTO report_error;
```

连接数据库

SQL 99

```
EXEC SQL CONNECT TO (db-name) [AS (connect-name)] [USER (username)]
```

Oracle

不需要数据库名

```
EXEC SQL CONNECT TO :user_name IDENTIFIED BY :user_pwd;
```

断开连接

SQL 99

在断开连接之前要 提交 或 回滚

```
EXEC SQL COMMIT work;
EXEC SQL ROLLBACK work;
```

```
EXEC SQL DISCONNECT (connect-name);
EXEC SQL DISCONNECT CURRENT;
```

Oracle

提交 或 回滚 后自动断开连接

```
EXEC SQL COMMIT release;
EXEC SQL ROLLBACK release;
```

C 语言中的 ESQL

先通过 Precompiler 把 ESQL 编译成 C 函数

游标

1.声明

```
EXEC SQL DECLARE agent_dollars CURSOR FOR
    select aid, sum(dollars)
    from orders
    where cid = :cust_id
    group by aid;
```

2.打开

- 在打开游标之前，要给变量 cust_id 赋值
- 在打开游标之后，游标的指针指向结果集的第一行之前

```
EXEC SQL OPEN agent_dollars;
```

3.取结果行

```
while (TRUE) {
    EXEC SQL FETCH agent_dollars INTO :agent_id, :dollar_sum;
    print(agent_id)
}
```

4.关闭游标

```
EXEC SQL CLOSE agent_dollars;
```

4.结束 fetch 循环

```
EXEC SQL WHENEVER NOT FOUND GOTO finish;
while (TRUE) {...}
finish:
    EXEC SQL CLOSE agent_dollars;
```

Whenever

```
EXEC SQL WHENEVER (condition) (action)
```

- condition
    - SQLERROR

        - 程序错误
    - NOTFOUND
        - 没有记录受到影响
    - SQLWARNING
- action
    - CONTINUE
    - GOTO (label)
    - STOP
    - DO function | BREAK | CONTINUE

指示符变量

- 0：一个非 0 的数据库值被赋给变量
- >0：一个截断的数据库字符串被赋给变量
- =-1：数据库值是 null，变量的值没有意义

```
/* 将 discnt 设为 null */
cd_ind = -1
EXEC SQL UPDATE customers
    SET discnt = :cust_discnt INDICATOR :cd_ind
    WHERE cid = :cust_id
```

事务

```
EXEC SQL SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

动态 SQL

三种类型

```
EXECUTE IMMEDIATE :sqlStentence;
```

```
/* 用 ? 做参数的占位符 */
PREPARE sqlSentence FROM :param
EXECUTE sqlSentence USING :param
```

数据库设计

常见问题

- 为什么把所有数据都放在一张表里不好？
- 数据冗余：大量重复数据
- 修改异常：修改某一行会要求其他行也被更改
- 删除异常：删除某一行会导致其他信息丢失
- 插入异常：没有全部的数据就无法执行插入

ER 模型

Entity-Relationship Model 实体关系模型

- 三个基本元素

    - 实体
    - 属性
        - 多值属性要有自己的表
    - 关系
        - E R F
        - 若 E 中某实体 $\rightarrow$ F 有多条连线，则 max-card(E, R) = N
            - 称为 **多值参与**
            - 若 = 1 称为 **单值参与**
        - 若 E 中每个实体都有到 F 的连线，则 min-card(E, R) = 1
            - 称为 **强制参与**
            - 若 = 0 称为 **选择参与**
        - 将 max-card = y 和 min-card = x 合并为 card(x, y)
            - x = 0 or 1
            - y = 1 or N
        - 一对一，多对一，多对多
            - 关系 $\rightarrow$ 表 的多个规则
- ER 图
    - 矩形：实体
        - 继承：鱼骨箭头指向父类
    - 菱形：关系
        - 菱形-矩形之间有 card(x, y)
    - 椭圆：属性
        - 矩形-椭圆之间有 (x, y)
            - (0, ?)：可选的
            - (1, ?)：强制的
            - (?, 1)：单值
            - (?, N)：多值

函数依赖

FD：Function Dependency

对于一个 A 有唯一的 B 与之对应，称为 A 函数决定 B，B 函数依赖于 A

Armstrong 公理

- Rule 1：若 X 包含 Y，则 X 函数决定 Y
- Rule 2：若 X 函数决定 Y 且 Y 函数决定 Z，则 X 函数决定 Z
- Rule 3：若 X 函数决定 Y，则 XZ 函数决定 YZ
- 推理
    - Rule 4：若 X 函数决定 Y 且 X 函数决定 Z，则 X 函数决定 YZ
    - Rule 5：若 X 函数决定 YZ，则 X 函数决定 Y 和 Z
    - Rule 6：若 X 函数决定 Y 且 WY 函数决定 Z，则 XW 函数决定 Z
    - Rule 7：若 X 函数决定 YZ 且 Z 函数决定 W，则 X 函数决定 YZW

闭包

- 函数依赖集 F 的闭包 记为 F^+

- F 中所有的函数依赖以及他们根据 Rule 推导出来的全部函数依赖构成了 F^+

覆盖

- 如果 G 中所有的函数依赖都可以从 F 中推导而来（即 G 被包含于 F^+），则称 F 覆盖 G
- 若 F 覆盖 G 且 G 覆盖 F 则他们等价

属性集的闭包

- $X_F^+ =\{ A|X \to A \in F^+ \}$
- 算法见 PPT

最小覆盖

- 最小的能够覆盖给定函数依赖集 F 的集合 M
  - 没有冗余的函数依赖
  - 每一个函数依赖的左边都没有多余的属性
- 算法
  - 创建一个 F 的等价集，右边都只有一个属性
  - 移除不重要的 FD
    - 不重要：如果移除 H 中的 $X \to Y$ 得到 J，有 $H^+ = J^+$ 或 $H = J$，则认为 $X \to Y$ 是不重要的
  - 反复地将 FD 替换为左边属性更少的 FD，只要结果不会改变 H^+
  - 把所有左边相同的 FD 根据 union rule 聚合起来
  - 创建一个全部左侧都不相同的依赖集 M

分解

- 将一张表分解成更多更小的表
- 总是能够重新获得全部原有行，但可能会得到更多

无损分解

- 若表 T 有函数依赖集 F，将 T 分成 T1~Tk 这 k 个表，有
  - 对于每一个 Ti，Head(Ti) 是 Head(T) 的子集
  - Head(T) = Head(T1) 并 ... 并 Head(Tk)
  - T 等价于 T1 join ... join Tk
- 对于表 T 的分解 { T1, T2 }，它是一个无损分解如果以下某个 FD 在 F 中
  - Head(T1) 交 Head(T2) $\to$ Head(T1)
  - Head(T1) 交 Head(T2) $\to$ Head(T2)

有损分解

- 同上
  - T 被包含于 T1 join ... join Tk

范式

依赖保持性

- 设关系 R 上的函数依赖集为 F，将 R 分解为 { T1,...Tk } 这 k 个子关系模式，从 F 中可以推导出在 Ti 上的函数依赖集为 Fi
- 若 F 和 F1 并 ... 并 Fk 是等价的，则称该分解具有依赖保持性

超键和键

- 定理 6.7.3：给定表 T ，函数依赖集 F，Head(T) 中某些属性的集合 X
  - X 是 T 的超键当且仅当 X 函数决定 T 中的全部属性
    - $X \to Head(T)$ 或 $X_F^+ = Head(T)$
- 求候选键的算法
  - 令 K = Head(T)
  - 对于 K 中的每个属性 A
    - 计算 $(K - A)_F^+$
    - 如果 $(K - A)_F^+$ 包含 T 中的全部属性
      - 则令 K = K - { A }

主属性

- 主属性是属于键的属性（不等同于主键）
- BCNF（Boyce-Codd Normal Form）
  - 一张表 T 符合 BCNF 当且仅当
    - 对 F^+ 中全部 $X \to A$（ X 和 A 中全部属性都属于 T），A 是单个属性且不在 X 中，则 X 必定是 T 的超键
- 3NF（第三范式）
  - 一张表 T 符合 3NF 当且仅当
    - 对 F^+ 中全部 $X \to A$，A 是单个非主属性且不在 X 中，则 X 必定是 T 的超键
- BCNF 和 3NF 关系：如果 T 符合 BCNF，则 T 符合 3NF
- 2NF（第二范式）
  - 一张表 T 符合 2NF 当且仅当
    - 对 F^+ 中全部 $X \to A$，A 是单个非主属性且不在 X 中，则 X 不包含在 T 的任意键中
- 产生符合 3NF 无损分解的算法
  - 将 F 替换为 F 的最小覆盖
  - S = 空集
  - 对 F 中 所有的 $X \to Y$
    - 如果对全部的 $Z \in S$，X 并 Y 不包含于 Z，则 S = S 并 Head(X 并 Y)
  - 如果，对 T 的全部候选键 K
    - 对全部的 $Z \in S$， K 不包含于 Z
    - 选择候选键 K，S = S 并 Head(K)

数据库管理

创建表

- 语法

```
CREATE TABLE customers (
    cid char(4) NOT NULL,
    cname varchar(13),
    PRIMARY KEY(cid)
)
```

- 一致性约束
  - 列约束
    - NOT NULL 或 DEFAULT NULL
    - 约束名（可选）
    - UNIQUE 且 NOT NULL：候选键
    - PRIMARY KEY
    - REFERENCES
      - FOREIGN KEY
        - 表 T1 中列的集合 F 被定义为 T1 的外键当且仅当 F 中任意一行的值结合起来都是 NULL 或是另一个表 T2 的候选键或主键
      - RESTRICT
      - CASCADE
      - SET NULL
    - CHECK

```
CREATE TABLE customers(
    cid char(4) NOT NULL,
    discnt real CONSTRAINT discnt_max
                CHECK(discnt <= 15.0)
)
CREATE TABLE orders(
    ordno integer NOT NULL,
    cid char(4) NOT NULL,
    dollars float DEFAULT 0.0 CONSTRAINT dollarsck CHECK (dollars >= 0.0),
    CONSTRAINT cidref FOREIGN KEY(cid) REFERENCES customers
    // 也可以直接用以下表示方法
    cid char(4) NOT NULL REFERENCES customers
)
CREATE TABLE emp(
    ssn char(8) PRIMARY KEY,
    // 若部门中有职工，不允许在 dept 表中删除该部门
    dno char(4) REFERENCES dept ON DELETE RESTRICT
    // 若部门中有职工，在 dept 中删除该部门时，在 emp 中删除该部门所有职工
    dno char(4) REFERENCED dept ON DELETE CASCADE
    // 若部门中有职工，在 dept 中删除该部门时，将职工 dno 设为 NULL
    dno char(4) REFERENCED dept ON DELETE SET NULL
)
```

修改表

- 触发器
  - 响应特定事件后自动执行的代码
  - 为了保证数据一致性
- 分类
  - 行级：某一行的任何一列值改变前 / 后触发
  - 列级：特定列改变前/ 后触发
  - For Each Row：结果集的每一行被影响了就执行一次
  - For Each Statement：对于整个结果集只触发一次

```
CREATE TRIGGER disnt_max
    AFTER INSERT ON customers
    REFERENCING NEW AS x
    FOR EACH ROW WHEN (x.discnt > 15.0)
    BEGIN
        raise_application_error(-20003, "invalid discnt");
    END;

CREATE TRIGGER foreign_cid
    AFTER DELETE ON customers
    REFERENCING OLD AS old_custom
    FOR EACH ROW
    BEGIN
        UPDATE orders
            SET cid = NULL
            WHERE cid = :old_custom.cid;
    END;
```

创建视图

- 通过子查询产生的表，但是有自己的表名和属性名
- 但是没有真正的数据储存，只是 SELECT 选取出来的数据（虚拟表）

```
CREATE VIEW agent_orders(ordno, cid, charge, aid, acity)
    AS SELECT o.ordno, o.cid, o.dollars, o.aid, a.aname
        FROM orders o, agents a
        WHERE o.aid = a.aid

// 视图可以用来做查询
SELECT sum(charge) FROM agent_orders WHERE acity = 'NJ'

// 不允许直接用同样的属性名
CREATE VIEW cacities  // 应改为cacities(ccity, acity)
    AS SELECT c.city, a.city
    FROM customers c, agents a
    WHERE c.cid = a.cid

// 带检查
CREATE VIEW custs
    AS SELECT * FROM customers WHERE discnt <= 15.0 WITH CHECK OPTION;
// UPDATE custs SET discnt = discnt + 4.0 对于 discnt 为 12.0 的行会失败吗？
```

- 一个视图是可修改的当且仅当子查询满足以下条件
    - FROM 子句必须只包含一个表 / 可修改视图
    - 不能出现 GROUP BY 或者 HAVING
    - DISTINCT 关键字不是特定的
    - WHERE 子句不包含引用了 FROM 子句中任何表的子查询
    - 子查询的结果列都是简单的列名，没有表达式，没有重复列名
- 在 ORACLE 中，你可以修改一个 join views 如果
    - join 是多对一的
    - 视图的列包含多对一那个多的表的主键
    - 而且只能修改多对一那个多的表的列

安全性

- 设定操作权限

```
GRANT SELECT, DELETE, INSERT, UPDATE(cname, city) ON custview TO hxw;
REVOKE(重设权限): 和 GRANT 语法相同，只是将 TO 改成 FROM
```