Chapter 6 Database Design

1. 数据依赖性与关系模式规范化 Data Dependency and Normalization of Relational Schema

1.1. 函数依赖 Function Depedency

1.1.1. 简介 Concepts

- 完全函数依赖（Fully Functional Dependency）：

  Y完全依赖于X，指没有任何额外的属性可以决定Y的值，只有X才能唯一确定Y的值

- 部分（函数）依赖（Partial Functional Dependency）

  Y依赖于X，但是Y的值可以被X的某个真子集所决定

1.1.2. 冗余的弊端 The Evils of Redundancy

- 冗余会导致与关系模式相关的几个问题：
  - 冗余存储、插入/删除/更新异常
- 完整性约束，特别是功能依赖关系，可用于识别存在此类问题的模式并提出改进建议。
- 主要改进技术：分解（将ABCD替换为AB和BCD，或ACD和ABD）

1.1.3. FD在检测冗余中的作用

- 考虑一个具有3个属性（ABC）的关系R
  - 没有FD：没有冗余
  - 给定A → B：几个元组可以有相同的A值，如果这样的话，它们都将具有相同的B值

1.1.4. 概念与特性

- 如果对于关系R函数依赖 $X \rightarrow Y$ 成立，意味着任意元组 t_1 和 t_2 ，如果 $t_1[x] = t_2[x]$ 则 $t_1[y] = t_2[y]$
- 函数依赖是关于所有允许的关系实例的声明，描述了关系中属性之间的依赖关系
  - 函数依赖的识别是基于应用领域的语义进行的，必须根据应用程序的语义来确定属性之间依赖关系

    - 给定关系R的某个实例 $r1$ ，可以检查是否符合函数依赖 f ，但不能判断 f 本身是否成立
- 如果 K 是关系 R 的候选键，意味着 $K \to R$ ，不要求 K 是最小的候选键
- 函数依赖允许我们表达无法使用超键表达的约束
    - 考虑以下模式： $in\_dep(ID, name, salary, dept\_name, building, budget)$
    - 期望满足函数依赖 $dept\_name \to building$ ，但不期望 $dept\_name \to salary$
- 平凡函数依赖（Trivial Functional Dependencies）
    - 定义：如果函数依赖 $\alpha \to \beta$ 中， $\beta \subseteq \alpha$ 的子集，那么这个函数依赖就是平凡的
    - e.g. $ID, names \to names$

1.1.5. 关于函数依赖（FDs）的推理

- 给定一些FD，我们通常可以推断出其他FD：
    - e.g. $ssn \to did, did \to lot$ 推出 $ssn \to lot$
- 一个FD f 被一组 FDs F 所推导出，意味着只要 F 中的所有FDs成立， f 也必定成立。
    - 我们可以使用 F^+ 来表示 F 的闭包。
- 阿姆斯特朗公理 *Armstrong's Axioms*（X、Y、Z是属性集）：
    - 自反性（Reflexivity）：若 $X \subseteq Y$ 则 $Y \to X$ 。
    - 增广性（Augmentation）：若 $X \to Y$ ，则 $\forall Z$， $XZ \to YZ$
    - 传递性（Transitivity）：若 $X \to Y, Y \to Z$ ，则 $X \to Z$
- 根据*Armstrong's Axioms*，推出附加规则：
    - 并集规则（Union）：若 $X \to Y, X \to Z$ ，则 $X \to YZ$
    - 分解规则（Decomposition）：若 $X \to YZ$ ，则 $X \to Y, X \to Z$
    - Example: Contracts(*cid, sid, jid, did, pid, qty, value*), and:
        - C is the key: C→CSJDPQV
        - Project purchases each part using single contract: JP → C
        - Dept purchases at most one part from a supplier: SD → P
    - JP → C, C → CSJDPQV imply JP → CSJDPQV SD→P implies SDJ→ JP
      SDJ → JP, JP→ CSJDPQV imply SDJ → CSJDPQV
- 计算一组函数依赖的 F^+ 很贵，因为大小与属性数量呈指数关系：只想检查 $X \to Y$ 是否在 F^+ 中：
    - 初始化 X^+ 为 $\{X\}$
    - 对于 F 中的每个函数依赖 $A \to B$ ，如果 A 包含在 X^+ 中，则将 B 添加到 X^+ 中
    - 重复步骤2，直到 X^+ 不再增大。=
    - 检查 X^+ 中有没有 Y

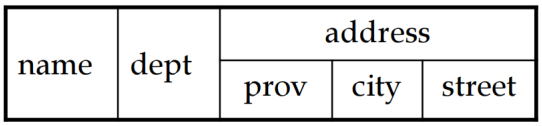
1.2. 规范化形式 Normal Forms（只要会判断）

1.2.1. 规范化形式的意义

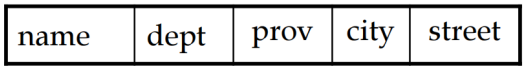
如果关系处于某种规范化形式（BCNF、3NF等），则已知某些问题被避免/最小化,这可以用来帮助我们决定分解关系是否会有所帮助

1.2.2. 第一范式（1NF）

- 定义：关系中的每一列属性都是不可再分解的原子属性



Non 1NF



1NF

1.2.3. 第二范式（2NF）：在1NF的基础之上，消除了非主属性对于候选键的部分函数依赖

- 定义：关系模式R属于1NF，并且属性之间不存在部分函数依赖
  - $S(S_{\#}, SNAME, AGE, ADDR, C_{\#}, GRADE)$ --- 非2NF
- 非2NF的问题：
  - 插入异常：e.g. 不能插入未选课的学生信息
  - 删除异常：e.g. 如果一个学生未选择所有课程，他的基本信息也会丢失
  - 更新困难：e.g. 由于冗余，在更新时很难保持一致性
- 解决方法：根据“one fact in one place”的规则将关系分解为两个新的关系：
  - $S(S_{\#}, SNAME, AGE, ADDR)$
  - $SC(S_{\#}, C_{\#}, GRADE)$

1.2.4. 第三范式（3NF）：在2NF基础上消除非主属性对于候选键的部分依赖与传递函数依赖

- 关系模式满足2NF且属性之间不存在传递函数依赖关系
  - $EMP\ (EMP_{\#}, SAL\_LEVEL, SALARY)$ --- 非3NF
- 如果对于关系R的 F^{+} 中的所有 $X \rightarrow A$ 满足以下条件之一时为3NF：
  - $A \subseteq X$（平凡函数依赖）
  - X 包含R的候选键（是超键）
  - A 中的每个属性都包含在一个候选键中 （是主属性）
- 无损联接（lossless-join）和依赖保持（dependency-preserving）的将关系分解成一组3NF关系总

是可能的，即可以将关系R分解为多个3NF关系，而不会丢失任何信息，并且保留了原始关系的所有函数依赖

- 非3NF的问题
  - 插入异常：e.g. 在决定员工工资级别之前，工资级别和工资之间的对应关系不能输入
  - 删除异常：e.g. 如果某个工资级别只有一个人，当这个人被删除时，这个级别的工资级别和工资之间的对应关系就会丢失
  - 难以更新：e.g, 由于冗余，很难在更新时保持一致性
- 解决方案：根据“一个事实在一个地方”的规则将关系分解为两个新的关系：
  - $EMP(EMP_{\#}, SAL\_LEVEL)$
  - $SAL(SAL\_LEVEL, SALARY)$

1.2.5. BCNF：在3NF基础上消除主属性对于候选键的部分与传递函数依赖

- 关系R的 F^+ 中的所有 $X \to A$ ，若满足以下条件之一，则R是BCNF：
  - 是平凡函数依赖
  - X 包含R的候选键（即X是超键）
- 换句话说，如果R中唯一的非平凡函数依赖是键约束，则R在BCNF中
  - 可以消除所有基于函数依赖能发现的冗余性
  - 如果我们看到两个元组在X值上达成一致，我们不能从其中一个元组的A值推断出另一个元组的A值
  - 如果关系实例在BCNF中，则两个元组必须是相同的（因为X是一个键）

1.2.6. 3NF 和 BCNF

3NF相对于BCNF来说是一种妥协

- 如果 $X \to A$ 违反了3NF，则有下列情况之一（A是非主属性）：
  - X 是某个候选键K的真子集
    - 是非主属性对候选键的部分函数依赖
  - X 不是任何候选键的真子集，也不是超键。
    - $K \to X \to A$ ，传递函数依赖
- 但是：即使关系模式处于3NF，也可能出现这些问题
  - 借船表：SBDC，S→C，C→S属于3NF，但对于每个船员S的预订情况，相同的(S, C)对会被存储

3NF优于BCNF的优势：

- 在不影响无损性和依赖性维护的前提下，总是可能得到一个3NF设计

3NF的缺点：

- 我们可能不得不使用空值来表示数据项之间的一些可能的有意义关系
- 存在信息重复的问题

1.2.7. 关系结构分离 Decomposition of a Relation Scheme

- 假设关系R包含属性A1 ... An。R的分解包括用两个或多个关系替换R，使得：
  - 每个新关系架构包含R的属性的子集（和不包含在R中的属性）
  - R的每个属性都作这些新关系中的数据属性出现
- 直观地说，R的分解意味着我们将存储由分解产生的关系方案实例，而不是R的实例。例如，可以将SNLRWH分解为SNLRH和RW
- 分解可能存在的潜在问题（权衡：必须权衡考虑这些问题与冗余性）：
  - 某些查询可能会变得更昂贵：多表查询
  - 考虑到分解关系中的实例，我们可能无法重建原始关系中的相应实例
  - 检查某些依赖关系可能需要将分解关系的实例进行联接。

1.2.8. 无损分解 Lossless Decomposition

- 假设R是一个关系模式，且R1和R2构成了R的一种分解。即R = R1 U R2
- 无损分解：分解后得到的关系表，自然连接以后依然等于原关系表 r（没有多出来行）
  - $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$
- 反之有损（多出了原本不存在的）
  - $r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$

1.2.9. 总结

- 如果一个关系符合BCNF范式，那么它就不会存在可以通过函数依赖检测到的冗余。因此，努力确保所有关系都符合BCNF是一个好的启发式方法
- 如果一个关系不符合BCNF，我们可以尝试将其分解为一组符合BCNF的关系
  - 在进行分解时，必须考虑是否保留了所有的函数依赖关系。如果无法进行无损联接和依赖保持的BCNF分解（或者考虑到典型查询的情况不适用），则应该考虑分解为3NF
  - 在进行分解或重新审查时，应该同时考虑性能要求

1.2.10. 补充一个例子

通常无法同时实现BCNF和依赖保留

- $dept\_advisor(s\_ID, i\_ID, dept\_name)$
- 具有函数依赖关系：
  - $i\_ID \to dept\_name$
  - $s\_ID, dept\_name \to i\_ID$
- dept_advisor不属于BCNF： i_ID 不是超键
- 对dept_advisor的任何分解都不包括 $s\_ID, dept\_name \to i\_ID$ 中的所有属性
- 因此无法实现BCNF和依赖保留。

但可以考虑3NF

- 两个候选键为： $\{s\_ID, dept\_name\}$ ，$\{s\_ID, i\_ID\}$
- $\{s\_ID, dept\_name\}$ 是一个超键
- $i\_ID \to dept\_name$ 同时 i_ID 不是超键，但
  - $\{dept\_name\} - \{i\_ID\} = \{dept\_name\}$ 同时 $dept\_name$ 在一个候选键中
  - 此处这么写的分析：
    - 在判断3NF时对于条件三还有一种写法是： $A - X$ 中的每个属性都包含在R的一个候选键中

1.3. 多值依赖（Multivalued Dependencies，MVD）

1.3.1. 定义

假设有一个关系模式 R ，其中 a 是 R 的一个属性集， b 是 R 的另一个属性集。如果在任何合法的关系 $r(R)$ 中，对于所有满足 $t_1[a] = t_2[a]$ 的元组对 t_1 和 t_2 ，存在元组 t_3 和 t_4 满足以下条件：

$$t_1[a] = t_2[a] = t_3[a] = t_4[a]$$

$$t_3[b] = t_1[b] \qquad t_3[R - b] = t_2[R - b]$$

$$t_4[b] = t_2[b] \qquad t_4[R - b] = t_1[R - b]$$

则有多值依赖 $a \to\to b$ 在R上成立，注意这个 R 不一定是完整的所有属性，可以是 $U \subseteq R$

1.3.2. 与函数依赖区别

- 函数依赖 $X \to Y$ 的有效性仅仅取决于 X ，Y 这两个属性集，不涉及第三个属性集
- 多值依赖中， $X \to\to Y$ 在属性集 $U(U = X + Y + Z)$ 上是否成立，不仅要检查属性集

X，Y 上的值，而且要检查属性集 U 的其余属性 Z 上的值。

1.3.3. 表格表示

	α	β	$R-\alpha-\beta$
t_1	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
t_2	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
t_3	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
t_4	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$

- *inst_info*(*ID*, *child_name*, *phone_number*)
- Example data:
  (99999, David, 512-555-1234)
  (99999, David, 512-555-4321)
  (99999, William, 512-555-1234)
  (99999, William, 512-555-4321)

1.3.4. 案例解读

- 在我们的例子中：
  - $ID \rightarrow\rightarrow child\_name$
  - $ID \rightarrow\rightarrow phone\_number$
- 上述正式定义是为了正式化这样一个概念，即给定 $Y(ID)$ 的特定值，它与 $Z(child\_name)$ 的一组值和 $W(phone\_number)$ 的一组值相关联，而且这两个集合在某种意义上彼此独立。
- 注意：
  - 如果 $Y \rightarrow Z$ ，那么 $Y \rightarrow\rightarrow Z$

2. 数据库设计方法 Database Design Method

- 以过程为中心的方法 Procedure oriented method

该方法以业务流程为中心，数据库模式的设计基本上直接根据业务中的凭证、收据、报告等来进行。由于没有对数据和数据之间的内在关系进行详细的分析，虽然该方法在项目初期很快捷，但很难保证软件质量，并且系统很难适应未来需求和环境的变化，因此这种方法不适合开发大型、复杂的系统。

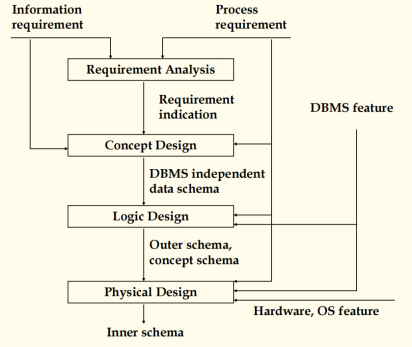
- 以数据为中心的方法 Data oriented method

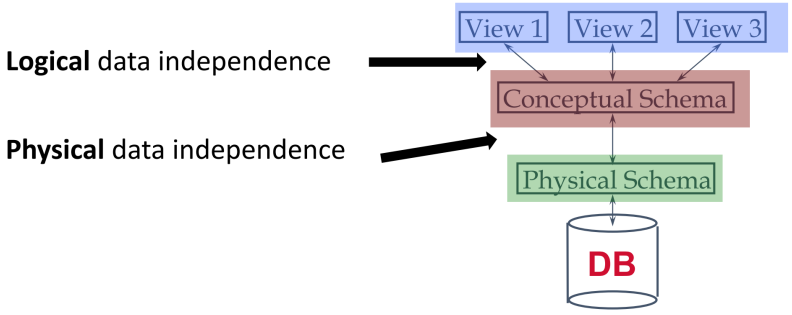
该方法基于对业务过程中涉及的数据和数据之间内在关系的详细分析来设计数据库模式。以数据为中心，而不是以过程为中心。不仅可以满足当前需求，还可以满足一些潜在需求。容易适应未来需求和环境的变化。在大型、复杂系统的开发中，我们推荐这种方法。

2.1. 数据库设计步骤

- 需求分析：用户需求；数据库需要完成哪些工作?
- 概念设计
  - 高层次描述（通常使用ER模型完成）

    - 对象关系映射（ORM：Hibernate，Rails，Django等）鼓励您在此处编程
- 逻辑设计
    - 将ER转换为数据库管理系统数据模型
    - ORM通常也需要您在此处提供帮助
- 模式精化
    - 一致性，规范化
- 物理设计 – 索引，磁盘布局
- 安全设计 – 谁访问了什么，以及如何访问。





2.2. 数据独立性

- 使应用程序与数据结构隔离
    - 逻辑数据独立性：当逻辑结构发生变化时，维护视图
    - 物理数据独立性：当物理结构发生变化时，维护逻辑结构。
- Q: 对DBMS来说为什么特别重要
    - 因为数据库及其相关应用程序会持久存在。

2.3. 需求分析

- 系统需求分析的一个非常重要的部分。在需求分析阶段，数据字典和DFD（或UML）图对数据库设计最重要。
- 字典和DFD图
    - 名称冲突、同义词（不同含义的同义词）、近义词（同义但在不同名称中）、概念冲突、领域冲突
- 关于编码
    - 信息标准化、识别实体、信息压缩
- **通过需求分析，所有信息都必须具有唯一的来源和责任**

2.4. 概念设计 Conceptual Design

基于数据字典和DFD，对数据字典中的数据进行分析和分类，并参考在DFD中反映的处理要求，识别实体、属性以及实体之间的关系，从而得到数据库的概念模式

- 识别实体
- 定义实体之间的关系
- 绘制ER图并与用户讨论

2.5. 逻辑设计 Logic Design

- 根据ER图中的实体和关系，定义目标DBMS中的表和视图。基本标准是3NF。
  - 将ER图中的实体和关系转换为表
  - 表的命名规则和属性命名规则
  - 定义每个属性的类型和域
  - 合适的反规范化
  - 必要的视图
  - 考虑遗留系统中的表
  - 接口表
- 将ER转换为关系型数据结构
  - 结构相当类似
  - 但是ER中的许多简单概念在关系型中很难精确地表达出来

Logical DB Design: ER to Relational

• Entity sets to tables. Easy.

ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

```
CREATE TABLE Employees
  (ssn CHAR(11),
   name CHAR(20),
   lot  INTEGER,
   PRIMARY KEY  (ssn))
```

Relationship Sets to Tables

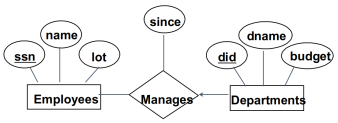
In translating a **many-to-many** relationship set to a relation, attributes of the relation must include:

1) Keys for each participating entity set (as foreign keys). This set of attributes forms a ***superkey*** for the relation.
2) All descriptive attributes.

ssn	did	since
123-22-3666	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92

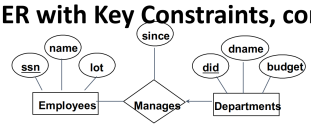
```
CREATE TABLE Works_In(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
    REFERENCES Employees,
  FOREIGN KEY (did)
    REFERENCES Departments)
```

Translating ER with Key Constraints




```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees,
 FOREIGN KEY (did)
   REFERENCES Departments)
```

Translating ER with Key Constraints, cont




```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees,
 FOREIGN KEY (did)
   REFERENCES Departments)
```

Vs.

```
CREATE TABLE  Dept_Mgr(
 did  INTEGER,
 dname  CHAR(20),
 budget  REAL,
 ssn  CHAR(11),
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
   REFERENCES Employees)
```

- Since each department has a unique manager, we could instead combine Manages and Departments.

Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - **When the owner entity is deleted, all owned weak entities must also be deleted.**

```
CREATE TABLE  Dep_Policy (
   pname  CHAR(20),
   age  INTEGER,
   cost  REAL,
   ssn  CHAR(11) NOT NULL,
   PRIMARY KEY  (pname, ssn),
   FOREIGN KEY  (ssn) REFERENCES Employees
      ON DELETE CASCADE)
```

2.6. 物理设计 Physical Design

- 对于关系数据库，此阶段的主要任务是根据处理要求考虑创建必要的索引，包括单属性索引、多属性索引、簇索引等。一般来说，经常作为查询条件的属性应该建立索引
- 其他问题：分区设计、存储过程、触发器、完整性约束