前言

本文章是根据法国国立高等电力技术、电子学、计算机、水力学与电信学校 (E.N.S.E.E.I.H.T.) 第八学期课程*"Base de Donnees"* 总结而来的【部分课程笔记】。碍于本人学识有限，部分叙述难免存在纰漏，请读者注意甄别。

数据库设计的步骤

1. 需求分析

   调查机构情况与熟悉业务活动，明确用户的需求，确定系统边界，生成用户字典和用户需求规格说明书

2. 概念结构设计

   将需求分析得到的用户需求抽象为概念模型，绘制E-R图

3. 逻辑结构设计

   将E-R图转换为与DBMS相符合的逻辑结构（包括数据库模式和外模式），例如E-R图向关系模型的转换，再根据规范化理论对数据模型进行优化，设计用户子模式。

4. 物理结构设计

   通常关系数据库物理设计的内容包括关系模式选择存取方法，以及设计关系、索引等数据库文件的物理存储结构

5. 数据库实施

   建立实际数据库结构、载入数据

6. 数据库运行与维护

数据库系统的标准结构

三级模式和两层映像

DBMS 管理数据的三个层次

1. `External Level` 或 `User Level`：某一用可以看到与处理的数据，全区数据中的某一部分
2. `Conceptual Level` 或 `Logic Level`：从全局的角度理解和管理的数据，含相应的关联约束
3. `Internal Level` 或 `Physical Level`：存储在物理介质上的数据

视图（View）与模式（Schema）

模式（Schema）：对数据库中的数据所进行的一种结构性的描述

视图（View）：某一种表现形式下表现出来的数据库中的数据

三个层次下的视图与模式

1. `External Schema` 与 `External View`：某一用户可以看到与处理的结构描述，及数据
2. `Conceptual Schema` 与 `Conceptual View`：从全局的角度理解和管理的数据的结构描述，含相应的关联约束。体现在数据之间的内在本质联系。
3. `Internal Schema` 与 `Internal View`：存储在物理介质上的数据的结构描述，含存储路径、存储方式、索引方式等

若只提到模式，则是指 `Conceptual Schema`；只提到视图时，则通常是指 `External View`。

两层映像

何为映像？将一种模式转化为另一种模式的过程称之为映像。

- `E-C Mapping` 将数据结构从 `Conceptual Schema` 转换到 `External Schema` 的映像，以便用户的观察和使用
- `C-I Mapping` 将数据结构从 `Internal Schema` 转换到 `Conceptual Schema` 的映像，以便计算机存储和处理数据

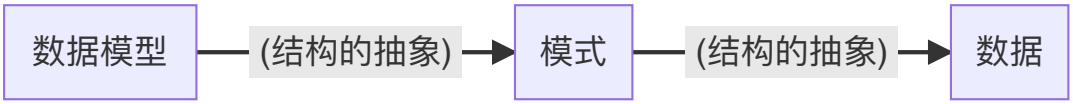
两个独立性

- 逻辑数据独立性：当 `Conceptual Schema` 变化时，可以不改变 `External Schema`，只需改变 `E-C Mapping`，从而无需改变应用程序。
- 物理数据独立性：当 `Internal Schema` 变化时，可以不改变 `Conceptual Schema`，只需改变 `E-C Mapping`，从而不改变 `External Schema`。

数据模型

前面我们提到了数据（视图）和模式，即模式是对数据本身的抽象。

而数据模型是对模式结构的抽象。



比如，关系模型：所有模式都可以抽象成表(Table)的形式，而每一个具体的模式都是具有不同列名的具体的表。

三大经典数据模型

- 关系模型 → 表的形式组织数据
- 层次模型 → 树的形式组织数据
- 网状模型 → 图的形式组织数据

数据库的关系模型

形象地说，一个关系就是一个表（但并不完全是），关系模型就是处理表的，通常由三部分组成：

1. 描述表的基本结构形式 (Relation/Table)
2. 描述表于表之间的各种关系运算 (`union, difference, product, selecet, project, intersect, join, division`)
3. 描述这些操作间应遵循的完整性约束条件 (实体完整性、参照完整性、用户自定义完整性)

什么是关系

1. 首先定义“列”的取值范围【域 (Domain)】

   $D = \{d_1, d_2, \dots, d_n\}$
2. 在定义“元组”及所有可能组成的 n元组 (来自不同域的值的排列组合)：笛卡尔积 (Cartesian Product)

   $D_1 \times D_2 \times \dots \times D_n = \{(d_1, d_2, \dots, d_n) \mid d_i \in D_i, i = 1, 2, \dots n\}$
   - 在元组 $\{(d_1, d_2, \dots, d_n)\}$ 的每一个值 d_i 叫做一个分量 (Conponent)
3. 关系：在笛卡尔积中所有的元组中一个“有意义”的子集
   - $\mathcal{R}(A_1 : D_1,\ A_2 : D_2,\ \dots,\ A_n : D_n) \subset \{D_1 \times D_2 \times \dots \times D_n\}$ 简写为 $\mathcal{R}(A_1,\ A_2,\ \dots,\ A_n)$

     ```
     // 关系模式
     Student(id char(8), name char(10), sex char(2), age integer, class char(6))
     // 关系值
     Student(12345678, 张三, 男, 20, 三班)
     ```

关系的特性

- 列是同质的：即每一个列中的分量来自同一个域，是同一个类型的数据；
- 不同的列可以来自同一个域，称其中的每一列为一个【属性】，不同属性要给予不同的属性名；
- 行/列位置可以互换
- 关系中任意两个元组不能相同，但在实际运用中，表中的记录可以重复
- 满足【第一范式】：属性不可再分

关系的一些重要概念

候选码 (Candidate Key)

关系中的一个属性组，其值能唯一标识一个元组，若从该属性组中去掉任何一个属性，他就不具有这种性质了，这样的属性组称作候选码。

> 在以下的关系模式中，
>
> ```
> Student(id, name, sex, age, class)
> ```
>
> 只有 `id` 是可以唯一确定一个元组的，其他的属性组合都不能唯一确定一个元组。所有属性 `id` 是 `student` 关系的一个候选码

> `ChoiceClasses(studentId, classId, studentName, className, grade)`
>
> 属性组 `(studentId, classId)` 可以唯一确定一个元组，所以该候选组是一个候选码

- 超码：能表示出所有属性的集合，候选码是最小的超码
- 主码：当出现多个候选码时，可以选定一个作为【主码 (Primary Key)】
- 主属性 (Prime Attribute)：包含在任何一个候选码中的属性是【主属性】，而其他属性是【非主属性】；
- 当候选码为全体属性时，我们称其为【全码 (All Key)】

外码 (Foreign Key)

关系 $\mathcal{R}_1$ 中一个非主属性组，在另一个关系 $\mathcal{R}_2$ 的主属性组中，则称这个属性组为关系 $\mathcal{R}_1$ 的【外码 (Foreign Key)】

- 两个关系 $\mathcal{R}_1$ 和 $\mathcal{R}_2$ 通常是靠外码连接起来的。

关系模型的完整性

1. 实体完整性

关系的主码的值不能为空（未知或无意义的值）

> 因为主码可以唯一确定一个元组，如果值为空（未知或无意义），我们就无法确定一个元组

2. 参照完整型

如果关系 $\mathcal{R}_1$ 的外码 K_F 与关系 $\mathcal{R}_2$ 的主码 K_P 相对应，则关系 $\mathcal{R}_1$ 中的每一个元组的 K_F 值等于关系 $\mathcal{R}_2$ 中每个元组的主码 K_P 值，或空值。

3. 用户自定义完整性

用户针对具体的应用环境定义的完整性约束条件

> 添加关于年龄的约束 $[0, 200]$

关系代数运算

基于集合，提供了一系列的关系代数操作：`union, difference, product, selecet, project, intersect, join, division` 等操作。以若干个关系为输入 $\rightarrow$ 结果是一个新关系。

并相容性

两个关系 $\mathcal{R}_1$ 和 $\mathcal{R}_2$ 存在相容性，当且仅当：

（1）关系 $\mathcal{R}_1$ 和 关系 $\mathcal{R}_2$ 的属性数目必须相同；

（2）对于任意 i，关系 $\mathcal{R}_1$ 的第 i 个属性的域必须和关系 $\mathcal{R}_2$ 的第 i 个属性的域相同

满足并相容性的两个关系可以进行并、交、差运算

并运算 (Union)

$$\mathcal{R}_1 \cup \mathcal{R}_2 = \{t \mid t \in \mathcal{R}_1 \vee t \in \mathcal{R}_2\}$$

其中，t 是元组。

差运算 (Difference)

$$\mathcal{R}_1 - \mathcal{R}_2 = \{t \mid t \in \mathcal{R}_1 \wedge t \notin \mathcal{R}_2\}$$

其中，t 是元组。

- $\mathcal{R}_1 - \mathcal{R}_2$ 与 $\mathcal{R}_2 - \mathcal{R}_1$ 是不同的

广义笛卡尔积 (Cartesian Product)

关系 $\mathcal{R}_1(\langle a_1, a_2, \ldots, a_n \rangle)$，关系 $\mathcal{R}_2(\langle b_1, b_2, \ldots, b_n \rangle)$，有

$$\mathcal{R}_1 \times \mathcal{R}_2 = \{\langle a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n \rangle \mid \langle a_1, a_2, \ldots, a_n \rangle \in \mathcal{R}_1 \wedge \langle b_1, b_2, \ldots, b_n \rangle \in \mathcal{R}_2\}$$

- $\mathcal{R}_1 \times \mathcal{R}_2 = \mathcal{R}_2 \times \mathcal{R}_1$

选择操作 (Select)

$$\sigma_{con}(\mathcal{R}) = \{t \mid t \in \mathcal{R} \wedge con(t) =' true'\}$$

- 设 $\mathcal{R}_1(A_1, A_2, \ldots, A_n)$，$t$ 是 $\mathcal{R}_1$ 的元组，t 的分量记为 $t[A_i]$ 或简写为 A_i
- 条件 con 由逻辑运算符连接几个比较表达式组成
- 逻辑表达式：$\wedge, \vee, \neg$
- 比较表达式：$X\ \theta\ Y, \theta \in \{>, \geq, <, \leq, =, \neq\}$
- 对于行操作（元组）

投影操作 (Project)

$$\Pi_{A_{i1}, A_{i2}, \ldots, A_{ik}}(\mathcal{R}) = \{\langle t[A_{i1}],\ t[A_{i2}],\ \ldots,\ t[A_{ik}] \mid t \in \mathcal{R}\}$$

- 设 $\mathcal{R}_1(A_1, A_2, \ldots, A_n)$，投影操作实际上就是从原有属性组中选出几个属性作为新的属性组
- $\{A_{i1}, A_{i2}, \ldots, A_{ik}\} \subseteq \{A_1, A_2, \ldots, A_n\}$
- 对于列操作（属性）

交操作 (Intersection)

$$\mathcal{R}_1 \cap \mathcal{R}_2 = \{t \mid t \in \mathcal{R}_1 \wedge t \in \mathcal{R}_2\}$$

- $\mathcal{R}_1 \cap \mathcal{R}_2 = \mathcal{R}_2 \cap \mathcal{R}_1$
- 交运算可以通过差运算来实现：$\mathcal{R}_1 \cap \mathcal{R}_2 = \mathcal{R}_1 - (\mathcal{R}_1 - \mathcal{R}_2) = \mathcal{R}_2 - (\mathcal{R}_2 - \mathcal{R}_1)$

θ - 连接操作 (θ - Join)

$$\mathcal{R}_1 \bowtie \mathcal{R}_2\ _{(A\ \theta\ B)} = \sigma_{(t_{[A]}\ \theta\ s_{[B]})}(\mathcal{R}_1 \times \mathcal{R}_2)$$

- A 是 $\mathcal{R}_1$ 的一个属性，设 $\mathcal{R}_1(A_1, A_2, \ldots, A_n)$，$A \in \{A_1, A_2, \ldots, A_n\}$
- B 是 $\mathcal{R}_2$ 的一个属性，设 $\mathcal{R}_2(B_1, B_2, \ldots, B_n)$，$B \in \{B_1, B_2, \ldots, B_n\}$
- θ 是比较表达式：$X\ \theta\ Y, \theta \in \{>, \geq, <, \leq, =, \neq\}$

- 即在不同的表中先求笛卡尔积，再在新表上进行 θ 操作

自然连接操作 (Join)

$$\mathcal{R}_1 \bowtie \mathcal{R}_2 = \sigma_{(t_{[B]}=s_{[B]})}(\mathcal{R}_1 \times \mathcal{R}_2)$$

- 关系 $\mathcal{R}_1$ 和 关系 $\mathcal{R}_2$ 必须有相同的属性组B

外连接 (Outer Join)

$$\begin{aligned} \mathcal{R}_1 ⟕ \mathcal{R}_2 \quad &左外连接 \\ \mathcal{R}_1 ⟖ \mathcal{R}_2 \quad &右外连接 \\ \mathcal{R}_1 ⟗ \mathcal{R}_2 \quad &全外连接 \end{aligned}$$

- 外连接 = 自然连接（θ - 连接）+ 失配的元组（与全空元组形成的连接）

除操作 (Division)

$$\mathcal{R}_1 \div \mathcal{R}_2 = \{t \mid t \in \Pi_{\mathcal{R}_1-\mathcal{R}_2}(\mathcal{R}_1) \wedge \forall u \in \mathcal{R}_2(tu \in \mathcal{R}_1)\}$$

- 常用越求解“查询全部”的问题

函数依赖

（1）非平凡的函数依赖：$X \to Y, Y \nsubseteq X$

> 无特殊说明下，均讨论非平凡的函数依赖。
>
> 意为 X 可以推导出 Y，且 Y 不是 X 的子集。

（2）平凡的函数依赖：$X \to Y, Y \subseteq X$

（3）完全函数依赖：$X \to Y$，并且对于 X 的任意真子集 X'，都有 $X' \nrightarrow Y$。则称 Y 完全范数依赖于 X，记作 $X \xrightarrow{F} Y$

> 意为：X 可以推导出 Y，且任意一个子集 X' 都不能推导出 Y，那么我们称 Y 完全范数依赖于 X

（4）部分函数依赖：Y 不完全范数依赖于 X，记作 $X \xrightarrow{P} Y$。

> 例如 $A \to C$ 且 $AB \to C$，我们可以看到，因为 A 已经可以推导出 C 了，对于 $AB \to C$ 来说，B 就是冗余的。

（5）传递函数依赖：Y 范数依赖于 X，Z 范数依赖于 Y，所以 Z 范数依赖于 X。
$X \to Y,\ Y \to Z \quad \Rightarrow \quad X \to Z$

范式

$$5NF \subset 4NF \subset BCNF \subset 3NF \subset 2NF \subset 1NF$$

$$1NF \xrightarrow{\text{消除【部分依赖】}} 2NF \xrightarrow{\text{消除【非主属性】的【传递依赖】}} 3NF \xrightarrow{\text{满足【箭头左侧搜包含候选码】}} BCNF$$

1NF

【第一范式】：要求所有属性都是不可分割的数据项（根据实际业务规定何为“不可分割”）。

第一范式的目标是确保每列（属性）的原子性

> 例如在具体的业务中，我们想记录用户的住址，但我们并不想对住址的省市加以区分。对于我们而言，地址就已经是一个“原子性的”数据项了。所以我们的第一范式就是：
>
> | id | 姓名 | 地址 |
> | --- | --- | --- |
>
> 而不是：
>
> | id | 姓名 | 省 | 市 | 区 | 详细地址 |
> | --- | --- | --- | --- | --- | --- |

第一范式可能存在的问题：由于模式中一些不正确的数据依赖关系所引起，可以分解关系来解决

1. 数据冗余
2. 更新异常
3. 插入异常
4. 删除异常

2NF

【第二范式】：在满足1NF的前提下，不包含非主属性对于候选码的部分函数依赖。即每个非主属性都是完全函数依赖于候选码的。

第二范式要求每个表只描述一类信息。

> 例如，在关系R中，候选码为【学生编号】和【教师编号】，非主属性是 [学生姓名]，[教师姓名]。因为通过【学生编号】就可以唯一确定 [学生姓名] 了，并不需要【教师编号】。所以，我们可以将1NF转化成2NF，即拆分成三个表：
>
> - 学生表(【学生编号】, 学生姓名)
> - 教师表(【教师编号】, 教师姓名)
> - 学生教师关系表(学生编号 , 教师编号)

3NF

【第三范式】：在满足2NF的前提下，不包含非主属性对于候选码的函数依赖。即候选码应该直接决定非主属性，不能间接决定 ($A \to B,\ B \to C$)

> 例如在关系R中，候选码为【客户姓名】，非主属性是 [订单编号] 和 [订单负责人]。则我们可以通过【客户姓名】决定 [订单编号]，再通过 [订单编号] 决定 [订单负责人]。所以，我们可以将其成3NF，即拆分成两个表：
>
> - 客户表(【客户姓名】, 订单编号)
> - 订单表(订单编号, 订单负责人)

BCNF

【BC范式】：在满足3NF的前提下，如果对于每个函数依赖 $X \rightarrow Y$，若 Y 不属于 X，则 X 必含有候选码，则该关系满足 $BCNF$。

SQL 语句

`DDL` 数据定义语句 (Schema)

`CREATE` 创建

关系名（表名）、属性名（列名）、属性的数据类型、完整性约束

```
CREATE TABLE '表名' (
  '列名' 数据类型 列的完整性约束 ,
  '列名' 数据类型 列的完整性约束 ,
  ... ,
) 表的的完整性约束;
```

`ALTER` 修改

```
ALTER TABLE '表名' (
  ADD COLUMN '新列名' 数据类型 列的完整性约束 ,
  ADD 完整性约束 ,
  MODIFY '列名' 数据类型 ,
  DROP COLUMN '列名' ,
  DROP 完整性约束 ,
)
```

`DROP` 删除

```
DROP TABLE '表名';
```

`DQL` 数据查询语言

`SELECT` 查询语句

```
-- 当满足_condition的条件下，从表T中查询列A
SELECT DISTINCT A as '列A的别名' -- DISTINCT 显示查询结果无重复
FROM T
WHERE _condition
```

子句：

- `WHERE`：条件子句，可以用 `AND / OR` 连接多个条件
  - 比较大小：`=, >, <, >=, <=, <>, !<, !>, NOT`
  - 确定范围 (包含)：`[NOT] BETWEEN ... AND...`
  - 指定集合：`列名 [NOT] IN (Val1, Val2, Val3)`，意为在查询结果中筛选列名的值为 `Val1, Val2, Val3` 的记录
  - 字符串匹配：`列名 [NOT] LIKE '匹配目标串' [ESCAPE '转义字符']`
    - `%` 通配符：代表任意长度的字符串。 `a%b` 表示以a开始，b结束的任意长度的字符串
    - `_` 通配符：代表单个字符。`a_b` 表示a开始，b结束的长度为3的字符串
  - 涉及空值：`IS [NOT] NULL`
- `GROUP BY`：对查询结果分组，

  分组方法：按指定的一列或多列值分组，值相等的为一组。
  - `HAVING`：功能类似于 `WHERE`，但是作用域是分组后的数据
- `ORDER BY`：对查询结果进行排序
  - `列名1,列名2 ASD`：升序（缺省）
  - `列名1,列名2 DESC`：降序
- 常用集函数：
  - `COUNT (列名)`：统计记录的个数（行数）
  - `SUM (列名)`：计算列总和
  - `AVG (列名)`：计算列平均值
  - `MAX (列名)`：计算列最大值
  - `MIN (列名)`：计算列最小值

`SELECT` 多表查询

同时涉及多个表的查询称为多表查询。

1. 交叉连接（广义笛卡尔积）：不常用

```
SELECT col1.*,col2.*
FROM col1,col2
-- 返回col1和col2的广义笛卡尔积
```

2. 等值连接

```
SELECT col1.*,col2.*
FROM col1,col2
WHERE col1.id = col2.id
-- 在col1和col2的广义笛卡尔积中找出id对应相等的结果
```

`SELECT` 嵌套查询

```
SELECT 列名
FROM 表名
WHERE _condition IN[>,<,=,...] (
  SELECT 内层列名
  FROM 内层表名
  WHERE _condition2)
```

- 内层查询的结果是一个关于“内层列名”的集合
- 不能使用 `ORDER BY` 和 `GROUP BY` 语句

`DML` 数据操作语言

`INSERT` 插入语句

```
-- 将指定元组加入指定表中
INSERT
INTO 表名 [(列名)]
VALUES val_1,val_2,...,val_n
-- 或子查询
```

`UPDATE` 修改语句

```
UPDATE 表名
SET 列名 = 表达式
WHERE _condition
```

`DELETE` 删除语句

```
UPDATE
FROM 表名
WHERE _condition
```

解题过程

1. 审题，列出所有的属性
2. 根据题目所述，写出所有的依赖关系“$X \to Y$”
3. 选出候选码：
    - 只出现在左边的一定是候选码
    - 只出现在右边的一定不是候选码
    - 左右两边都出现的可能是候选码
    - 左右两边都不出现的一定是候选码
    - 再确定候选码的闭包，如果可以推出全部集合，那么当前确定的就是候选码；否则就需要把每一个可能的值加入候选码中求解闭包，判断能否推出全部集合。

      【如何求闭包？】

      即可以由当前属性组的所有子集推出的，包含属性组中所有属性的属性组集合

      $$(A,B)^+ = \{(A \to \dots) \cup (B \to \dots) \cup (AB \to \dots) \cup (A) \cup (B)\}$$
4. 判断当前关系属于哪一个范式，并将其转化成BCNF

to be continued...