1. 概论

数据库的历史

- 人工；
- 文件系统；
- 数据库：
  - 1960年代末：层次性、网络；
  - 1970年代：关系模型；
  - 现在：OR （面向对象关系模型）, XML。

Database 数据库: a collection of data.

- very large amounts of data （海量的数据）；
- Structured and interrelated data （结构化的信息以及信息之间的联系）；
- operational data （与运转相关的信息）；
- persistent data （持久保存——**数据库只能建在磁盘上**）.

数据库通过 **DBMS (Database Management System)** 建立和管理（可以看成系统软件）。

通过 applications 应用程序使用数据。

为什么要数据库?

- 数据共享 information sharing；
- 数据独立性 data independence：
  - 一个程序 + 一个数据 = 数据非独立 （P1 + D1, P2 + D2, ...）；
  - 多个程序 + 一个数据库 = 数据独立（于程序） ((P1, P2, ..., Pn) + D）。

DBMS 数据库管理软件

- 需要持久存储海量数据；
- 需要能够高效访问数据。

DBS 数据库系统

- 硬件 hardware；
- 数据库 database；
- 数据库管理系统 DBMS；
- 用户（DBA 数据库管理员, applications programmer 应用程序员, end user 终端用户）

2. 关系数据模型

数据模型：提供了一套概念性的描述框架（工具）。**三要素**：

- 数据结构（数学表示）（例如，关系模型的数据结构即为关系 relation / table）；
- 对数据的操作；
- 对数据的约束。

一些数据模型

- 今天常用的模型：关系模型（relational），对象关系模型（object relational, OR）、半结构化数据模型（semistructured）；其中现在常用关系模型，半结构化模型更加灵活，但是较少使用。
- 其他模型：纯面向对象模型（OO）；
- 历史模型：层次模型、网络模型。

关系模型的特点

- 简单 simple，但是有一定限制 limited，万能的 versatile；
- 提供了有限但是有用的一系列操作；
- 允许利用 SQL 语言进行实现；
    - 使用非常简单：只需要陈述需要什么；
    - 效率稍显低下：但是可以通过一些优化改善。

关系：二维表格。属性 attribute (columns, headers) / 元组 tuple (rows)

> 数学上的关系：笛卡尔积的子集。

关系模式 (relation schema)：关系名与属性集合。

- 写作 *relation_name (attribute_list)*；实质上属性是无序的集合 set，并不是有序的表 list；但是一般使用时看成有序（定义的次序）；
- 可以把每一个属性的类型和其他信息添加进去；
- 一般来说，不随着时间而变化。
- **数据库模式**：数据库中存在许多关系，因此数据库模式即为关系模式的集合。

关系实例 (relation instance)：是元组的集合，可以随着时间改变，是一个无序的集合，且数据库仅维护当前的实例。

- **数据库实例**：关系实例的集合。

定义域 (domain)：属性的原子参数（类型）；

- 可以在关系模式中加入，例如 R(A1: D1, A2: D2)（D1、D2 为定义域）。

键值 (key)：一系列能够区分不同元组的属性（没有两个元组有着相同的键值）；

- 可以在关系模式中表示，例如 R(A1, A2, A3)（A1、A2 位键值）；
- 实质上，键值是一种语义上的约束；
- 有时候，键值需要使用人造信息（artificial keys）；
- 关系模式中kennel有许多种键值的选择方式。

为什么选用关系模型?

- 非常简单的模型；
- 非常高层次的编程语言：SQL，简单但是强大；
- 设计理论：数学上是严谨的。

SQL：标准的数据库语言。

- DDL (data definition language) + DML (data manipulate language)
- **SQL 中的关系**：

    - **stored relation**：利用表 table 来存储；
    - **view**：不存储，按需建立（临时计算）；
    - **temporary table**：通过 SQL 处理器建立。
- SQL 的数据类型：CHAR，BIT，BOOLEAN，……
    - SQL 2016 可以自定义类型。
- SQL 的值
    - 用单引号来表示字符串，字符串内存在单引号时，利用两个单引号表示；
    - 任意值可以为空 （NULL）；
    - 日期与时间：
        - 日期：`DATE 'yyyy-mm-dd'`；例如：`DATE '2011-01-11'`；
        - 时间：`TIME 'hh:mm:ss'`；例如：`TIME '15:30:02.5'`。
- 定义关系模式：

```
CREATE TABLE name (
     column_1     type_1,
      ... ,
     column_n     type_n
);
```

  例：

```
CREATE TABLE Student (
     sno CHAR(10),
     name VARCHAR(20),
     age INTEGER,
     dept VARCHAR(30)
);
```

    - `CHAR` 是定长的字符类型（上例中，定义 `CHAR(10)` 表示字符长度固定为 10 字节，不足 10 字节在末尾补空）；
    - `VARCHAR` 是变长的字符类型（上例中，定义 `VARCHAR(30)` 表示字符长度最长为 30 字节）。
- 删除关系模式：

```
DROP TABLE relation;
```

- 更改关系模式：

```
ALTER TABLE relation
ADD column type;
ALTER TABLE relation
DROP column;
```

- 缺省值（如果不定义，默认为 NULL）：

```
CREATE TABLE Student (
     sno CHAR(10),
     name VARCHAR(20),
     age INT DEFAULT 18,
     dept VARCHAR(30) DEFAULT 'CS'
);
```

- 定义键值
  - 单参数键值：利用 PRIMARY KEY；一个模式中，最多仅有一个主要键值。

    ```
    CREATE TABLE Student (
          sno CHAR(10) PRIMARY KEY,
          name VARCHAR(20),
          age INTEGER,
          dept VARCHAR(30)
    );
    ```

  - 多参数键值：将多个属性组成一个 PRIMARY KEY。

    ```
    CREATE TABLE SC (
          sno CHAR(10),
          cno CHAR(5),
          grade INTEGER,
          PRIMARY KEY (sno, cno)
    );
    ```

> - UNIQUE 与 PRIMARY KEY 的差别与联系：
>   - 他们都声明了这个属性的唯一性；
>   - 一个关系模式只有一个 PRIMARY KEY，但可能有许多 UNIQUE 属性；
>   - PRIMARY KEY 不能为空 （NULL），但是 UNIQUE 可以为空 （NULL）；
>   - UNIQUE 中，可以出现多个 NULL 值。

3. 关系代数

代数：利用运算符 (operator) 和运算对象 (operands) 构成表达式 (expression) 即为代数。

关系代数：

- *运算对象*：关系（可以用关系名字或元组的集合表示）；
- *运算符*：集合的运算与特殊的关系运算符。
- *查询* (query)：关系代数的表达式。

集合操作（令两个操作的关系分别为 R 和 S）

- 并 (union)：$R \cup S$，需要在插入后去重。
  - ***SQL 中，交、并、差默认进行去重操作，可使用 `ALL` 不去重提高效率。***
- 交 (intersection)：$R \cap S$。

- 差 (difference)：$R-S$，相当于删除。
- **条件**：两个操作的关系必须具有**相同的模式**。值得注意的是，$R(A,B)$ 和 $S(B,A)$ 本质上具有相同的模式，但操作时需要将属性排序后再进行集合操作。

> 【例】数据库中有三个关系 `Student(sno, name, age, dept)`，`Course(cno, name, credit)`，`SC(sno, cno, grade)`。
>
> - 关系本身可以看成一个查询：查询 `Student` 将会返回 `Student`。
> - 这个例子在下文中会不断使用。

投影 (projection) $\pi_L(R)$：只取某些属性列得到的结果表。

- L 是 R 的一个属性的列表，例如 $\pi_{name,age}(Student)$。
- 同样需要进行去重操作，例如 $\pi_{age}(Student)$。

选择 (Selection) $\sigma_C(R)$：只取某些行得到的结果表，其中 C 是一个包含 R 中属性的条件表达式（可以包含属性，常数，=，<，>，AND，OR，NOT 等等）。例如 $\sigma_{age>20 \text{ AND } dept='CS'}(R)$。

- 如何计算？将 C 应用于关系的每一行，判断是否符合条件表达式，若是则加入结果表。

笛卡尔积 (Cartesian Product) $R\times S$：结果表的属性是两个关系属性的并集，将两个关系的***所有可能配对组***加入结果表（关系表 R 的所有元组与关系表 S 的所有元组配对得到的所有可能配对组）。

- 如果关系具有相同的属性名，可以采用全名进行区分，如 `Student.name` 与 `Course.name`；
- 例如，$Student\times Course$，$R\times R$；
- 由于配对不相关的元组意义不大，所以笛卡尔积不常用。

条件连接 (Theta Join) $R\bowtie_\theta S$：将满足特定条件的配对组加入结果表，实际上即为 $\sigma_\theta(R\times S)$（用 θ 限制可能的匹配）。

- 匹配成功的元组称为 joined tuple，否则称为 dangling tuple；
- 例如，$\pi_{name}(Student\bowtie_{Student.sno=SC.sno \text{ AND } age>grade} SC)$；
- 更多的情况下，条件 θ 会涉及 R 与 S 的公共属性。

自然连接 (Natural Join) $R\bowtie S$：特殊的条件连接，θ 被隐式指定为 R 与 S 共有的属性完全相同。

- 结果表中，相同的属性只保留一个；
- 例如 $\pi_{Student.name,Course.name,grade}(Student\bowtie SC\bowtie Course)$。

商 (quotient) $T=R\div S$：定义在 $R(X,Y)$ 和 $S(Y)$ 上，得到商 $T(X)$；商定义为

$$T=\{t\mid t\in\pi_X(R) \text{ AND } (\forall s\in S)(ts\in R)\}$$

> 【例】
>
> 找到选了所有课的学生：$\pi_{sno,cno}(SC)\div\pi_{cno}(Course)$；
>
> 找到至少选择了所有 007 号同学选的课的学生：$\pi_{sno,cno}(SC)\div\pi_{cno}(\sigma_{sno='007'}(SC))$。

半连接 (semijoin) $R\ltimes S$：R 中与 S 中的至少一个元组的共有属性完全相同的元组组成的表，即为 $\pi_R(R\bowtie S)$。

换名 (naming and renaming)：控制关系/属性名。如 $\rho_S(R)$，$\rho_{S(A_1,A_2,\cdots,A_n)}(R)$，$\rho_S(expression)$，$\rho_{S(A_1,A_2,\ldots,A_n)}(expression)$ 等等。

查询 (query)：将操作进行组合得到查询。

- 如果两个表达式得到相同的结果，则称为等价表达式；可以用查询优化器优化。
- 运算符优先集：$(\pi, \sigma, \rho) > (\times, \bowtie_\theta, \bowtie) > (\cup, \cap, -)$，或直接使用括号。

表达式树 (expression tree)：用查询的表达式构建出表达式树。

启发式规则：将 σ 操作在表达式树中尽量下沉可以提高效率：

> 【例】
>
> $\pi_{sno,grade}(\sigma_{dept='CS'}(Student) \bowtie SC)$ 与 $\pi_{sno,grade}(\sigma_{dept='CS'}(Student \bowtie SC))$ 得到相同的结果，但是前者效率更高。

运算的独立集：$\{\cup, -, \pi, \sigma, \times, \rho\}$，其余运算可以用这些运算表示出来。

赋值 (assignment) $R(\cdots) := expression$：中间变量的存储。

> 【例】
>
> $R(no, n, a, d) := \sigma_{dept='CS'}(Student)$
>
> $Answer(sno, name) := \pi_{no,n}(R)$

引用完整性约束 (referential integrity constraints)：被引用的数据必须存在。

> 【例】在 `SC(sno, cno, grade)` 中，如果有一个元组 `('001', 'CS123', 90)`，那么 `001` 必须在 `Student` 中存在，`'CS123'` 必须在 `Course` 中存在。我们将这个约束表示为：
>
> $$\pi_{cno}(SC) \subseteq \pi_{cno}(Course)$$

键值约束 (key constraints)：某些参数不能出现相同的值。

> 【例】限制不存在两个相同的学生：$\sigma_{S_1.sno=S_2.sno \text{ AND } S_1.dept \neq S_2.dept}(S_1 \times S_2) = \varnothing$，其中 $\rho_{S_1}(Student), \rho_{S_2}(Student)$。

一些其他约束

- 添加定义域限制，如 $\sigma_{age<0 \text{ OR } age>200} = \varnothing$；
- 更复杂的限制，如 $\sigma_{age<18 \text{ AND } cno='C1'}(Student \bowtie SC) = \varnothing$。