

1. 数据

能够输入计算机并能被计算机程序识别和处理的符号集合

2. 数据库管理技术发展四阶段

人工管理阶段(1950)、文件系统阶段(1950-1969)、数据库系统阶段(1970)、多数据库技术阶段(1990-)

3. 数据库(Database/DB)

是存储在计算机内的有组织、可共享的数据集合，可以看成是电子文件柜

4. 数据库中的数据满足

- ①数据按一定的**数据模型**组织，在描述和存储上具有较小的冗余度
- ②数据之间相互**联系**
- ③由**数据库管理系统(DBMS)**进行管理

5. 数据模型及三要素

• 数据的组织方式，不但表达数据本身，而且表达数据之间的联系，以及数据的一致性约束

①数据结构：组织方式 ②数据操作：对数据进行的各种操作及规则

③数据约束：数据本身及数据之间存在的依存及制约关系

• 历史上出现过的数据模型：**层次、网状、关系**、面向对象、对象关系、XML

6. 数据库管理系统

• 定义：对数据进行统一管理和控制的软件系统（介于用户和 OS 之间的系统软件，基于某数据模型）

• 目标：科学地组织和存储数据、高效地获取和维护数据。

• 功能：数据库的①定义功能②操纵功能③运行控制功能

7. 数据库系统 DBS

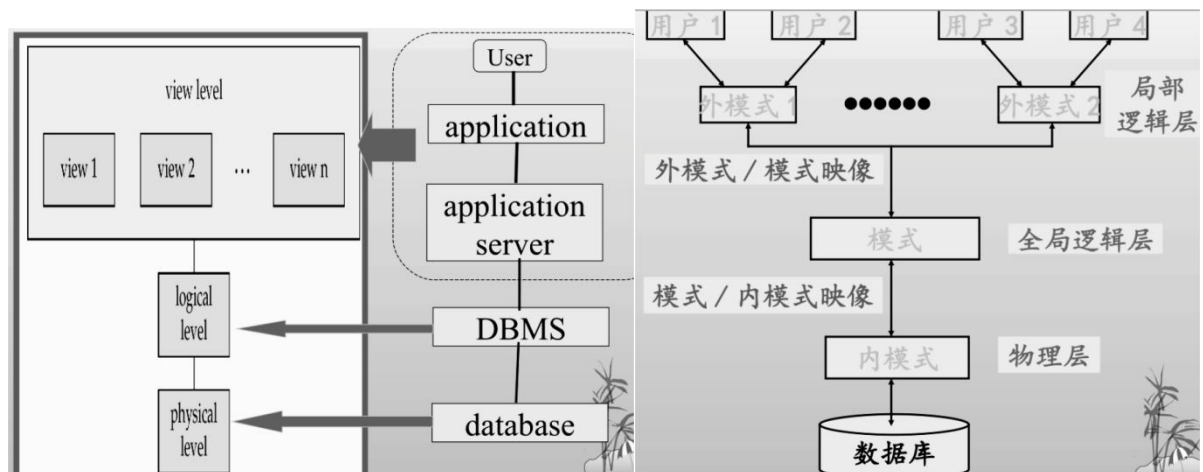
是 DB、DBMS、应用程序、DBA(数据库管理员)、计算机系统平台的总称

8. DBS 的三级结构

• 视图层：描述用户使用的数据结构

• 逻辑层：描述数据库的逻辑结构

• 物理层：①数据存储②存储路径③所占空间④填充率⑤存储块大小⑥记录存储⑦索引⑧结构⑨存取



• 数据独立性：数据与应用程序在一定程度上相互分离

• 逻辑独立性：模式**改变**->外模式/模式映像**改变**->外模式**不变**->应用程序**不变**

• 物理独立性：内模式**改变**->模式/内模式映像**改变**->模式(尽量)**不变**->外模式和应用程序**不变**

9. DBA

对数据库系统（应用程序和数据）进行集中控制的人，他的职责如下：

- ①**模式(schema)定义**
- ②存取方式定义
- ③模式的修改
- ④用户授权
- ⑤日常维护

10. 文件系统&数据库系统

• 文件系统的优点①长期保存②物理&逻辑结构有简单区别③数据不再属于某程序

• 文件系统的缺点①**没统一数据模型，数据访问困难**②数据孤立③完整性问题④**更新的原子性问题**⑤并发异常⑥安全性问题

• 相比之下**数据库系统的优势**：①数据共享②减少不必要的冗余③较高数据独立性④加强数据安全性

第二章 关系模型

1. 关系模型

用二维表（集合）来组织数据及数据之间联系的模型

- (1) 属性：每一列必须有名字，称为属性
- (2) 域：属性的取值集合
- (3) 元组：每一行数据。相当于一个记录值
- (4) 关系的特点
 - ①关系表中每一个属性都是原子的
 - ②各属性不能重名
 - ③行列次序不重要，交换行列不影响语义
 - ④不允许相同元组出现
- (5) 关系模式（instances）与关系实例（schemas）如右
 - ①关系名与关系的属性集称为**关系模式（模式）**
 - ②对应一个模式的当前元组集合，称为**关系实例（关系）**
 - ③**模式相对稳定，关系经常变动**
- (6) 键码

定义：若关系中的某一属性（组），它的值能唯一的标识一个元组，且它任意一个**真子集不能标识**，那么就称他为关系的**键码/候选码/码/主码**

特点：**唯一性，最小性**

★相关重要概念★

- ①超码：以键码为**子集**的任意属性集
 - ②**主属性：构成键码的属性**
 - ③**非主属性：不构成键码的属性**
 - ④外码：关系 R 中的属性 X 是另一个关系 S 的键码，则 X 是 R 的外码（foreign key）。是表示关系之间联系的重要手段
- (7) 关系表示数据

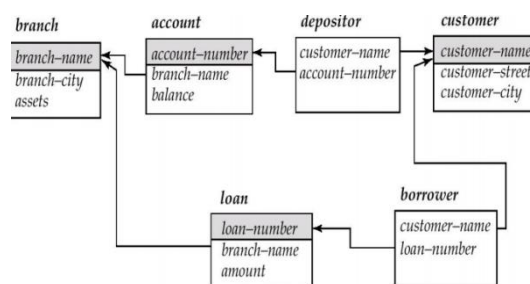
用关系模型组织数据时，按数据的特征分类，用一张二维表组织

- (8) 关系表示数据间联系

用关系模型表示两类数据的联系时，将能代表两类数据属性和描述联系性质的属性组成一张表

(9) 关系数据库

是若干关系构成的集合，一般来说，数据库中的数据组织方式用模式图来表示



2. 关系代数

是**抽象的、过程化的查询语言**，特点是：运算**对象**和运算**结果**都是**关系**

(1) 基本运算

(1). 传统的集合运算:

并 (UNION; \cup)

差 (DIFFERENCE; $-$)

交 (INTERSECTION; \cap)

笛卡尔乘积 (CARTESIAN PRODUCT; \times)

(2). 关系运算:

选择 (SELECT; σ)

投影 (PROJECTION; π)

改名运算 (一元运算; ρ)

连接运算: (JOIN; \bowtie)

并运算: 记作 $R \cup S$

差运算: 记作 $R - S$

交运算: 记作 $R \cap S$

关系R和S必须是并兼容的!

(2) 并兼容问题：两个关系的属性个数相同，并且对应属性的域也相同，则称这两个关系并兼容。

(3) 选择运算、投影、改名

从某一关系R中选出满足条件表达式F的元组，构成一个新关系，记为 $\sigma_F(R)$ 。

■ F是条件表达式，由比较及逻辑运算符构成。

\wedge (and), \vee (or), \neg (not)

$=, \neq, >, \geq, <, \leq$

■ Example of selection:

$\sigma_{branch-name='Perryridge'}(account)$

从某一关系R中选出某些属性A，构成新的关系，记为 $\pi_A(R)$ 。

■ E.g. $\pi_{account-number, balance}(account)$

■ 在投影的结果里，重复的元组将被删除

把一个关系的名称改成另外一个名称。一般来说用于对关系运算的结果进行命名。记为：

$\rho_s(R)$

$\rho_s(A_1, \dots, A_n)(R)$ ：把关系R的名字改为S，并且把S的属性依次命名为 A_1, \dots, A_n 。

$\rho_s(customer)$

(4) 笛卡尔乘积、 θ 连接、自然连接

两个属性个数分别是 m 、 n ，元组个数分别为 k_1 、 k_2 的关系 R 和 S ，它们的笛卡尔乘积是一个关系，该关系属性个数(度/元)为 $m + n$ 、元组个数为 $k_1 \times k_2$ ，记为 $R \times S$

例	R (A, B, C)	S (A, BB)
	<hr/>	<hr/>
	a 1 c	aa 11
	b 2 f	bb 22

	R×S (R, A,	B,	C,	S, A,	BB)
→	a	1	c	aa	11
	a	1	c	bb	22
	b	2	f	aa	11
	b	2	f	bb	22

两个关系的元组在满足某种条件下进行匹配，即两个关系在公共属性上进行等值连接的运算。
θ-连接。记为： $R \bowtie_{\theta} S$

<i>E. g.</i>	$R1 \ (A1, \ A2)$	$R2 \ (B1, \ B2)$
	$a1 \ 1$	$b1 \ 1$
	$a2 \ 2$	$b2 \ 1$
	$a3 \ 1$	$b3 \ 2$
	$a4 \ 2$	$b4 \ 3$

$$R1 \bowtie R2 = (A1, A2, B1, B2)$$

$A2 > B2$	$a2$	2	$b1$	1
	$a2$	2	$b2$	1
	$a4$	2	$b1$	1
	$a4$	2	$b2$	1

记为: $R \bowtie S$

E. g. $R(A, B) \bowtie R(S) = (A, B, C)$

$a1$	1	$a1$	1	$b1$
$a2$	1	$a1$	1	$b2$
$a2$	2	$a2$	1	$b1$
$a3$	1	$a2$	1	$b2$
S (B, C)		$a2$	2	$b2$
1	$b1$	$a3$	1	$b1$
1	$b2$	$a3$	1	$b2$
2	$b2$			

(5) 复合运算：注意灵活运用逻辑运算

eg.从 Movies(title,year,length,studio)关系里面选出由 Fox 公司制作的长度不少于 100 分钟的电影的名称和年份。 $\Pi_{\text{title,year}}(\sigma_{\text{studio} = \text{"Fox"} \wedge \text{length} \geq 100}(\text{Movies}))$

3. 关系的完整性

- ①实体完整性：保证关系中的每一个元组都是可识别且唯一的。
- ②参照完整性：一个表中的一个元组的存在以另一个表中某一元组的存在为前提，外码实现
- ③用户定义的完整性/域完整性/语义完整性：针对某一具体应用领域定义对数据的约束条件
- 作用：执行插入、删除、更新操作时检查完整性

第三章 数据库建模与实体联系模型

问题（目的）：如何将现实世界中的数据组织到数据库中

1. 数据库设计的基本步骤

- (1) 内容: 构造较优的数据模式
- (2) 步骤: 需求分析、设计概念模型、设计逻辑模型、设计物理结构、系统实现、试行&维护
- (3) 概念模型: 从应用语义视角来抽取模型, 按用户的观点来对数据和联系进行建模
特点是面向用户、面向现实世界, 与 DBMS 无关
- (4) 表达工具: **实体-联系模型 (E-R)**, 语义对象模型

2. 实体-联系模型 E-R

由实体和联系构成的模型

(1) 基本概念

- ①实体：现实世界中**客观存在**的一个具体或抽象的事物。能与其他实体**相互区分**。
 - 属性：**用来描述实体特征的数据项**。
 - 实体标识符：能够唯一标识一个实体的属性（组），且它的子集不能标识。
- ②实体集：具有共同属性的一类实体的集合。
 - 属性类型：简单属性、组合属性、单值属性、多值属性、派生属性（由其他属性计算而来）
- ③联系：实体之间的相互关系
 - 联系的属性：描述联系的特征
 - 一个联系集的所有联系有相同的属性
- ④联系集：两个或多个实体集间，实体间的联系的总集合
 - 在关系模型中，联系集可以用关系来表达：
 $\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$ 其中 E_i 表示实体集 (e_1, e_2, \dots, e_n) 是一个元组
 - Example: depositor 表示的是客户与帐户间的联系集, $(\text{Hayes}, A-102) \in \text{depositor}$
 - 联系的度：一个联系集所涉及的实体集个数
 - 涉及两个实体集的联系，叫**二元联系**
 - 三个及以上，叫多元联系

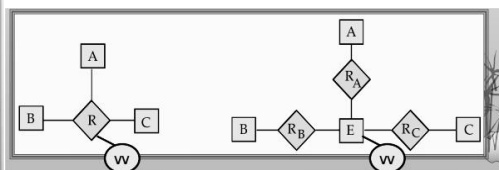
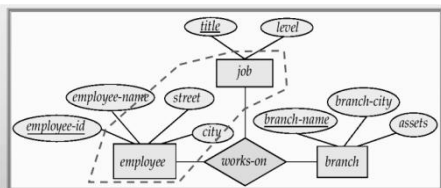
(2) E-R 图基本概念

- ### ①基本概念:

- ②联系中的角色 **roles**: 如果联系涉及的实体集是一个, 需要区分它在联系中的不同作用

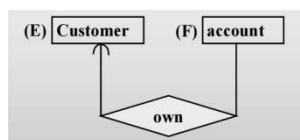
```

    erDiagram
        employee ||--o{ employee : works-for
        employee {
            string employee-name
            string employee-id
            string telephone-number
        }
        works-for {
            string manager
            string worker
        }
  
```



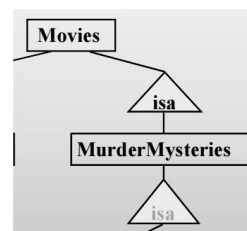
- ④多元联系转化为二元联系:将R用实体集E来代替,然后生成以E与其它实体集间的二元联系:RA, RB, RC原联系R的属性成为E的属性

④选择合适的事物类型

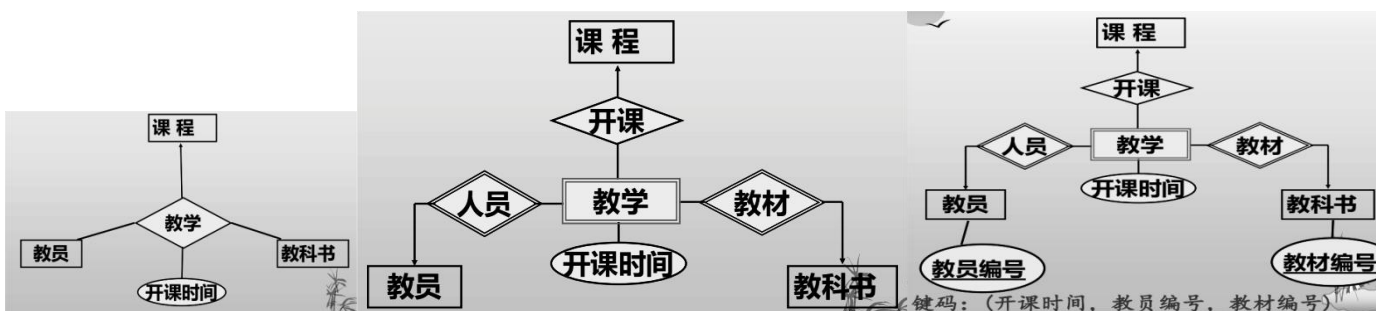


③域约束：属性类型和取值范围的约束。在 E-R 图中没有特殊规定

④当子类实体没有自己的属性和联系时，则其不出现在 E-R 图中!!!!



②多远联系转化的连接实体集往往是弱实体集



- ④弱实体集与为它提供键码的实体集间的联系用双线棱形

③子类实体集转化:

method1: 父类主码 + 子类自己的属性

method2: 子类继承的所有属性 + 自己拥有的属性

(2) 联系集转化

• 原则: 出现相同属性名应该改名

• 关系的键码由联系的类型决定

①多对多: 联系自己的属性 + 参与联系的实体集的键码 (这些实体集的键码都作为新键码)

②一对多

• 联系自己的属性 + “多”实体集的键码 (“多”的键码为新键码)

• 可以不用单独转化, 只需要在“多”的实体集属性中, 加入联系的属性和“一”的键码

③弱实体集与它所依赖的强实体集间的联系没有必要转化 (“多”的键码为新键码)

④一对一

• 联系自己的属性 + 双“一”的键码 (任选一个作为新键码)

• 可以不用单独转化, 合并两个“一”的所有属性, 键码从原来各自的键码任选一个

第四章 关系数据库设计理论

1. 函数依赖 ★★重点掌握★★ 函数依赖最高到 BCNF!!!

1. 函数依赖: 设 $R(U)$ 是属性集 U 上的关系模式, $X, Y \subseteq U$ 。如果对于 $R(U)$ 的任意一个关系 r , 以及 r 的任意两个元组 t_1, t_2 , 不存在: $t_1[x]=t_2[x]$, 而 $t_1[y] \neq t_2[y]$, 则称 X 函数决定 Y , 或者说 Y 函数依赖于 X 。记为: $X \rightarrow Y$ 。

注: 1) “ $X \rightarrow Y$ ”必须对 $R(U)$ 的任何一个关系实例都成立。

2) 若 $X \rightarrow Y, Y \rightarrow X$, 则记作 $X \leftrightarrow Y$ 。若 Y 不函数依赖于 X , 则记作 $X \nrightarrow Y$ 。

■ K is a candidate key (候选码) for R if and only if

1) $K \rightarrow R$, and

2) for no $\alpha \subset K, \alpha \rightarrow R$

■ Example4. $R=(SN, CN, GRADE)$

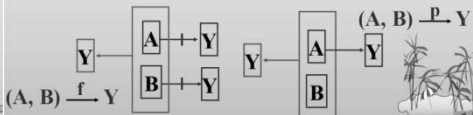
1) $(SN, CN) \rightarrow GRADE$

2) $SN \nrightarrow GRADE$

3) $CN \nrightarrow GRADE$

1) 完全函数依赖: 在 $R(U)$ 中, 如果 $X \rightarrow Y$, 且对 X 的任何一个真子集 X' , 都有 $X' \nrightarrow Y$, 则称 Y 对 X 完全函数依赖, 记作: $X \twoheadrightarrow Y$ 。

2) 部份函数依赖: 1) 的条件下, 如果存在 $X' \rightarrow Y$, 则称 Y 对 X 部份函数依赖, 记作 $X \twoheadrightarrow Y$ 。



Example5. $R=(sno, city, zip, pno, qty)$

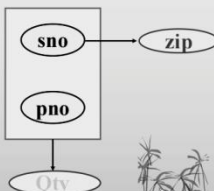
1) $(sno, pno) \rightarrow qty$

2) $(sno, pno) \twoheadrightarrow qty$

3) $(sno, pno) \rightarrow zip$

4) $sno \rightarrow zip$

5) $(sno, pno) \twoheadrightarrow zip$



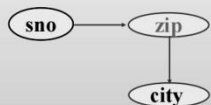
3. 传递函数依赖: 在 $R(U)$ 中, 如果 $X \rightarrow Y, Y \twoheadrightarrow Z$, ($Y \nsubseteq X$), $Y \rightarrow Z$, 则称 Z 传递函数依赖于 X , 记为: $X \twoheadrightarrow Z$ 。

■ Example6. $R=(sno, city, zip, pno, qty)$

1) $sno \rightarrow zip$

2) $zip \rightarrow city$

3) $sno \twoheadrightarrow city$



4. 非平凡的函数依赖: 若 $X \rightarrow Y$, 且 $Y \nsubseteq X$, 则称 $X \rightarrow Y$ 是非平凡的函数依赖。

■ Example7. $R=(sno, city, zip, pno, qty)$

1) $(sno, pno) \rightarrow qty$ 非平凡的函数依赖

2) $(sno, pno) \rightarrow pno$ 平凡的函数依赖

2. 范式

(1) 定义: 关系数据库中所有关系要满足的属性间依赖的要求

(2) 1NF: 各属性域必须是原子的

(3) 2NF: R 是一个关系模式, 若 $R \in 1NF$, 且每一个非主属性完全函数依赖于主码

Example8. $R=(sno, city, zip, pno, qty)$

1) $KEY=(sno, pno)$

2) \because 存在 $(sno, pno) \twoheadrightarrow city, zip$

3) $\therefore R \notin 2NF$

■ $S(sno, city, zip)$, $KEY=(sno)$

■ $SP(sno, pno, qty)$, $KEY=(sno, pno)$

■ $S \in 2NF, SP \in 2NF$ 。

■ 因而, 整个关系数据库模式也是 2NF 的。

(4) 3NF: $R \in 2NF$, 且不存在键码 X 、属性组 Y 以及非主属性 Z (Z 不包含于 Y) 使得 $X \rightarrow Y$ (且反之不成立), $Y \rightarrow Z$ 成立, 则 $R \in 3NF$

人话: R 中不存在非主属性对主码的传递函数依赖

■ Example9. $S(sno, city, zip)$, $KEY=(sno)$

■ $S \in 2NF$

■ S 不是 3NF。



■ $S(sno, zip)$, $KEY=(sno)$

■ $ZC(city, zip)$ $KEY=(zip)$

■ S, ZC 均属于 3NF。

a. 所有非主属性完全依赖于每个键码。

b. 所有主属性完全依赖于不包含它的键码。

c. 没有任何 (主/非主) 属性完全函数依赖于任一非主属性 (组)。

(5) BCNF: $R \in 1NF$, 对于每一个 $X \rightarrow Y$ (Y 不属于 X), X 是超码。三个特点如上图 ↑

3. 多值依赖与 4NF

(1) 多值依赖 MVD: R 是属性集 U 上的一个关系模式, X, Y 和 Z 都是 U 的子集, 并且 $Z = U - X - Y$ 。多值依赖 $X \twoheadrightarrow Y$ 成立, 当且仅当对于关系模式 R 的任意一个关系 r , r 在 X 上的一个值对应唯一一组 Y 值, 且这组 Y 值与 Z 的值无关。比如一门课可以对应唯一一组老师, 和该课用哪本教材无关。

(2) 结论

①对称性: $X \twoheadrightarrow Y$, 则必有 $Y \twoheadrightarrow X$, 当且仅当 $Z = U - X - Y$

②若 $X \twoheadrightarrow Y$, 当 $Y \subseteq X$ 或 $Z = U - X - Y = \Phi$, 则称为 $X \twoheadrightarrow Y$ 为平凡的多值依赖。否则就是非平凡的 MVD

③传递性: 若 $X \twoheadrightarrow Y, Y \twoheadrightarrow Z (Z = U - X - Y)$ 则 $X \twoheadrightarrow Z$ 。

(3) 4NF: $R \in 1NF$, 如果对于 R 的每个非平凡 $X \twoheadrightarrow Y$, X 都是超码, 则 $R \in 4NF$ 。

(4) 理解: 非平凡多值依赖的一种约束!!!!, 将每个非平凡的多值依赖左端限制为超码, 实际上说明这个多值依赖是函数依赖!



4. 函数依赖集闭包

(1) 函数依赖的逻辑蕴含: F 是 R 给定的一组函数依赖集合, 若能用逻辑推理导出其他函数依赖, F 逻辑蕴含这个导出的依赖 If $F = \{A \rightarrow B, B \rightarrow C\}$, then we can infer that $A \rightarrow C$ $F \Rightarrow A \rightarrow C$

(2) 闭包: F 及由 F 所蕴含的所有的函数数据依赖的集合, 记作 F^+

★★★★下面开始都是重点★★★★

5. 函数依赖 Armstrong 公理系统和推理规则

合并: 若 $X \rightarrow Y, X \rightarrow Z$, 则 $X \rightarrow YZ$
 分解: 若 $X \rightarrow YZ$, 则 $X \rightarrow Y, X \rightarrow Z$
 伪增广: 若 $X \rightarrow Y, W \supseteq Z$, 有 $XW \rightarrow YZ$
 伪传递: 由 $X \rightarrow Y, WY \rightarrow Z$, 有 $WX \rightarrow Z$ 。

Example 15. $R = (A, B, C, G, H, I)$
 $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
 some members of F^+

- $A \rightarrow H$
by transitivity from $A \rightarrow B$ and $B \rightarrow H$
- $AG \rightarrow I$
by augmenting $A \rightarrow C$ with G , to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
- $CG \rightarrow HI$
from $CG \rightarrow H$ and $CG \rightarrow I$
- $AG \rightarrow H$
From $A \rightarrow C$ and $CG \rightarrow H$

6. 属性集 A 关于函数依赖集 F 的闭包运算

- F 为属性集 U 上给定的一组函数依赖, $a, \beta \subseteq U$ 。
- 则 a_F^+ 称为属性集 a 关于函数依赖集 F 的闭包。
- $a_F^+ = \{\beta \mid a \rightarrow \beta \text{ 是所有由 } F \text{ 给定和导出的函数依赖集合}\}$ 。
- Example 16. $R(A, B, C), F = \{A \rightarrow B, B \rightarrow C\}$
 $\Rightarrow A_F^+ = \{ABC\}$

Example 17. $R = (A, B, C, G, H, I)$
 $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
 $(AG)^+$
 PROCEDURE:

- result = AG
- Result = $ABCG$ ($A \rightarrow C$ and $A \rightarrow B$)
- result = $ABCGH$ ($CG \rightarrow H$ and $CG \subseteq ABCG$)
- result = $ABCGHI$ ($CG \rightarrow I$ and $CG \subseteq ABCGH$)

- (1) 确定关系模式的键码
- 如何确定属性组 AG 是键码 (Is AG a candidate key)?
 - Does $AG \rightarrow R$? \Rightarrow Is $(AG)^+ \supseteq R$
 - Is any subset of AG a key?
 - 1. Does $A \rightarrow R$? \Rightarrow Is $(A)^+ \supseteq R$
 - 2. Does $G \rightarrow R$? \Rightarrow Is $(G)^+ \supseteq R$

Example 19. $R = (C, G, S)$
 $F = \{CS \rightarrow G\}$, Find all candidate keys for R

$(C)^+ = C, (G)^+ = G, (S)^+ = S$
 $(CG)^+ = CG, (CS)^+ = CSG, (SG)^+ = SG$

$CS \rightarrow R$
 STOP!
 CS is a key of R !!!!

7. 键码求解, 依赖集等价计算

(1) 键码求解

- ①只出现在函数依赖左边的属性, 必定是构成键码的属性;
- ②只出现在函数依赖右边的属性, 必定不是构成键码的属性;
- ③不出现在函数依赖中的属性, 也必定是构成键码的属性;
- ④既出现在左边, 也出现在右边的属性则可能是构成键码的属性。

(2) 依赖集等价

如果 $F^+ = G^+$, 就说函数依赖集 F 和 G 相互覆盖, 或 F 与 G 等价, 记为 $F=G$ 。

$F=G$ 的充分必要条件是 $F \subseteq G^+$, 且 $G \subseteq F^+$

判定: $F=G$?

判断 F 中的每一个依赖是否属于 G^+

判断 G 中的每一个依赖是否属于 F^+

例1: 给定关系模式 $R(X, Y, Z, U, V, W)$

, 依赖集 $F=\{X \rightarrow Z, XY \rightarrow Z, Z \rightarrow UW, ZV \rightarrow XY, ZU \rightarrow W, VW \rightarrow Z\}$, 求: R 的所有键码?

答案: R 的键码有两个: XV 和 ZV

Example 21. $F = \{A \rightarrow B, B \rightarrow C, AB \rightarrow C\}$,

$G = \{A \rightarrow B, B \rightarrow C\}$

$F = G$? $AB \rightarrow C \in G^+?$? ? ?

$AB^+_G = ABC$

$C \in AB^+_G$ 有 $F \subseteq G^+$

同理有: $G \subseteq F^+$

$F=G$ 成立!!

8. 模式分解

把一个关系模式 R 分解成若干个不相包含的关系模式 $\{R_1, R_2, \dots, R_n\}$ 的操作

9. 关系投影&无损连接性判断

(1) 投影举例 Decomposition of $R = (Eno, Es, Eg)$ into $R_1 = (Eno, Es)$ and $R_2 = (Eno, Eg)$

(2) 无损连接 $R = R_1 \cup R_2$, $r = \Pi R_1(r) \Join \Pi R_2(r)$

(3) 判断——分解成两个或多个子模式

例: 一个分解具有无损连接性的例子。

设: 关系模式 $R(Eno, Es, Eg)$ 。

$F = \{Eno \rightarrow Es, Es \rightarrow Eg\}$

现有分解: $\rho = \{R_1(Eno, Es), R_2(Es, Eg)\}$,

问: 该分解具有无损连接性吗?

易见, $(Es \rightarrow Eg) \in F^+$

根据定理, 分解 ρ 具备无损连接性。

定理: 设有关系模式 $R(U, F)$, 以及 R 的一个分解

$\rho = \{R_1(U_1, F_1), R_2(U_2, F_2)\}$ 。

如果 $(U_1 \cap U_2) \rightarrow (U_1 - U_2) \in F^+$, 或者:

$(U_1 \cap U_2) \rightarrow (U_2 - U_1) \in F^+$ 。

则, 分解 ρ 是具有无损连接性的。

例: 设有关系模式 $R = (U, F)$ 。其中:

$U = \{A, B, C, D, E\}$

$F = \{A \rightarrow D, E \rightarrow D, D \rightarrow B, BC \rightarrow D, CD \rightarrow A\}$ 。

现有 R 的一个分解:

$\rho = \{R_1(A, B), R_2(A, E), R_3(C, E),$

$R_4(B, C, D), R_5(A, C)\}$ 。

解: (i) 初始化: 创建5行5列的矩阵, 并赋初值

	A	B	C	D	E
R ₁	a1	a2	b13	b14	b15
R ₂	a1	b22	b23	b24	a5
R ₃	b31	b32	a3	b34	a5
R ₄	b41	a2	a3	a4	b45
R ₅	a1	b52	a3	b54	b55

$F = \{A \rightarrow D, E \rightarrow D, D \rightarrow B, BC \rightarrow D, CD \rightarrow A\}$ 。

	A	B	C	D	E
R ₁	a1	a2	b13	b14	b15
R ₂	a1	b22	b23	b14	a5
R ₃	b31	b32	a3	b14	a5
R ₄	b41	a2	a3	a4	b45
R ₅	a1	b52	a3	b14	b55

(ii) 根据函数依赖集 F , 反复检查、修改矩阵元

, 直到 F 中任何一个函数依赖 $X \rightarrow Y$, 矩阵元取值都不改变。

根据 $A \rightarrow D$ 检查、修改矩阵元。

根据 $E \rightarrow D$ 检查、修改矩阵元。

$F = \{A \rightarrow D, E \rightarrow D, D \rightarrow B, BC \rightarrow D, CD \rightarrow A\}$ 。

	A	B	C	D	E
R ₁	a1	a2	b13	a4	b15
R ₂	a1	a2	b23	a4	a5
R ₃	a1	a2	a3	a4	a5
R ₄	a1	a2	a3	a4	b45
R ₅	a1	a2	a3	a4	b55

(iii) 结论:

矩形的第三行元素为 $a1, a2, a3, a4, a5$, 它都是由形为 $a_i (i=1, \dots, m)$ 的元素构成, 所以 R 的分解是无损连接性的。

算法要点:

①反复使用函数依赖集, 直到矩阵元都不改变。

②当某列上的一个 b_{ij} 改为 $b_{kj} (k < j)$ 或 a_j 时, 应该把同一列、其它行上所有取值为 b_{ij} 的矩阵元都要改成 b_{kj} 或 a_j 。

③使用函数依赖集检查、修改矩阵元之后, 若矩阵上存在一行, 它都是由形为 $a_i (i=1, \dots, m)$ 的元素构成, 则分解是无损连接性的; 否则分解不具有无损连接性

10. 依赖的投影

E. g. $R(A, B, C), F = \{A \rightarrow B, B \rightarrow C\}$

$\rho_1(U_1, F_1)$, 其中 $U_1 = \{B, C\}$ 。问: $F_1 = ?$

$\rho_2(U_2, F_2)$, $U_2 = \{A, B\}$ 问: 此时 $F_2 = ?$

F_i 的算法:

$\rho_i(U_i, F_i)$ 是 R 分解得到的一个子模式, A 是 U_i 的一个真子集, 即 $A \subset U_i$

对于所有 $A \subset U_i$, 计算 A^+_F 和 $A^+_F \cap U_i$

根据上面的结果构造 F_i

Example 24. $R(U, F), U = \{A, B, C, D\}$,

$F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}, \rho = \{R_1(A, B, C),$

$R_2(C, D)\}$, 求: F 在 R_1, R_2 上的投影 F_1, F_2

求解过程:

$U_1 = ABC, U_2 = CD$

For $U_1, A^+_F, B^+_F, C^+_F, AB^+_F, AC^+_F, BC^+_F$

$A^+_F = ABCD, A^+_F \cap U_1 = ABC \Rightarrow A \rightarrow B, A \rightarrow C$

$B^+_F = ABCD, B^+_F \cap U_1 = ABC \Rightarrow B \rightarrow A, B \rightarrow C$

$C^+_F = ABCD, C^+_F \cap U_1 = ABC \Rightarrow C \rightarrow A, C \rightarrow B$

$AB^+_F = ABCD, AB^+_F \cap U_1 = ABC \Rightarrow AB \rightarrow C$

$AC^+_F = ABCD, AC^+_F \cap U_1 = ABC \Rightarrow AC \rightarrow B$

$BC^+_F = ABCD, BC^+_F \cap U_1 = ABC \Rightarrow BC \rightarrow A$

$F_1 = \{A \rightarrow B, A \rightarrow C, B \rightarrow A, B \rightarrow C, C \rightarrow A,$

$C \rightarrow B, AB \rightarrow C, AC \rightarrow B, BC \rightarrow A\}$

$F_2 = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

11. 保持函数依赖的分解

(1) 概念 $(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$

(2) 一些结论

①R 一定可以无损连接性地分解成若干个 BCNF(甚至是 4NF) 的子模式

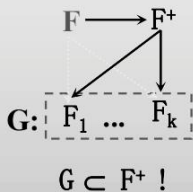
②R 一定可以保持函数依赖性地分解成若干个 3NF 的子模式, 但不一定能达到 BCNF。

③R 可以既保持函数依赖性, 又实现无损连接性地分解形成一些 3NF 的子模式, 但不一定能分解而形成 BCNF 的子模式。

2. 判定保持函数依赖性的算法

(1) 计算 $G = \bigcup_{i=1}^k F_i$

(2) 若 $F \subseteq G^+$,
则 ρ 是保持函数依赖性的。



例: 设: $R(U, F)$, $U = \{A, B, C, D\}$,
 $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$,
 $\rho = \{R_1(A, B, C), R_2(C, D)\}$.
问: ρ 是否保持函数依赖?

解: 得到: $F_1 = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

$F_2 = \{C \rightarrow D, D \rightarrow C\}$.

(1) 计算: $G = F_1 \cup F_2$
 $= \{A \rightarrow B, B \rightarrow C, C \rightarrow A, C \rightarrow D, D \rightarrow C\}$

(2) 对于 $D \rightarrow A \in F$, $\therefore D^+_G = \{ABCD\} \therefore A \subseteq D^+_G$ 即:
 $D \rightarrow A \in G^+$; 而对于 F 中的 $A \rightarrow B$, $B \rightarrow C$ 和 $C \rightarrow D$, 它们显然属于 G^+ .

故: ρ 对 R 的分解是保持函数依赖性的。

12. 保持依赖分解到 BCNF

算法: $R(U, F)$

① 初始, 令 $\rho = \{R(U, F)\}$

② 重复以下步骤, 直到 ρ 的所有关系模式都是 BCNF 的:

③ 找出 ρ 中不是 BCNF 的关系模式 $R_i(U_i, F_i)$, 其中必有: $X \rightarrow Y \in F_i$, 而 X 不含 R_i 的键码。

④ 计算: $X_{F_i}^+$, 并令: $A = X_{F_i}^+ - X$

⑤ 把 R_i 分解成子模式 R_{i1} 和 R_{i2} , 其中:
 $R_{i1} = X_{F_i}^+$, $R_{i2} = U_i - A$

⑥ 计算 F_i 在 R_{i1} 和 R_{i2} 的投影, 以及 R_{i1} 和 R_{i2} 的键码。

Example 25: $R = \{C, T, H, R, S, G\}$

$F = \{CS \rightarrow G, C \rightarrow T, HR \rightarrow C, HS \rightarrow R, TH \rightarrow R\}$

$KEY = \{HS\}$

$\rho = \{R\}$

$\rho = \{R_1, R_2\}$

$\rho = R$

$F = \{CS \rightarrow G, C \rightarrow T, HR \rightarrow C, HS \rightarrow R, TH \rightarrow R\}$

$KEY = HS$

$(CS)^+_F = \{CSGT\}$

$R_1 = CSGT$

$F_1 = \{C \rightarrow T, CS \rightarrow G\}$

$KEY = CS$

$R_2 = CHRS$

$F_2 = \{CH \rightarrow R, HR \rightarrow C, HS \rightarrow R, HS \rightarrow C\}$

$KEY = HS$

$\rho = \{R_1, R_2\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

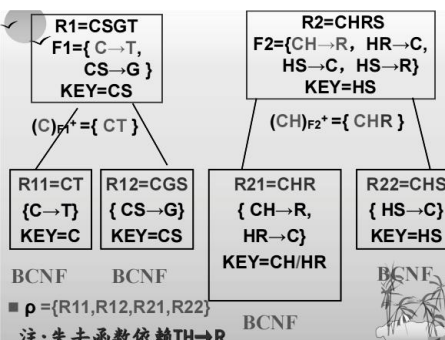
$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$

$\rho = \{R_1, R_2, R_3, R_4\}$



13. 保持依赖分解到 3NF

算法: $R(U, F)$

① 计算 F 的正则覆盖 F_m 。

② 若 F_m 中仅有一个函数依赖 $X \rightarrow Y$, 且 $XY = U$, 则
 $\rho = \{R(U, F_m)\}$, 并结束算法, 否则:

③ 把 U 中与 F_m 的所有函数依赖的左部和右部都无关的属性, 从 U 中分离出去, 并使它们构成一个关系模式 R_0 (R_0 至少是 3NF 的!)。设 F_m 剩余的属性构成属性集 U_0'

④ 按照 F_m 中各函数依赖的左部属性, 把 F_m 划分为 F_1, \dots, F_k , 使得每个 F_i ($i=1, 2, \dots, k$) 的所有函数依赖的左部都相同。

⑤ 根据 F_i 所包含的属性构造 U_i 。并由此构造关系模式 $R_i(U_i, F_i)$ 。于是, 可得 R 的一个分解 $\rho = \{R_0, R_1, \dots, R_k\}$, 且有: $U^k = \bigcup_{i=1}^k U_i = U_0'$

⑥ 对于 $1 \leq i, j \leq k$ ($i \neq j$), 若 $U_i \subseteq U_j$, 则把 R_i “合并”到 R_j 。

Example 26: $R = \{Eno, En, Es, Eg\}$

$F = \{Eno \rightarrow EnEsEg, En \rightarrow Eno, Es \rightarrow Eg\}$

① 求 $F_m = \{Eno \rightarrow En, Eno \rightarrow Es, En \rightarrow Eno, Es \rightarrow Eg\}$

② 所有的属性在 F_m 中均有! 按决定因子划分 F_m

$F_1 = \{Eno \rightarrow En, Eno \rightarrow Es\}$

$F_2 = \{En \rightarrow Eno\}, F_3 = \{Es \rightarrow Eg\}$

$R_1 = \{Eno, En, Es\}, R_2 = \{En, Eno\}, R_3 = \{Es, Eg\}$

$R_2 \subseteq R_1, F_1 = \{Eno \rightarrow EnEs, En \rightarrow Eno\}$

③ 最后的模式集合是: $R_1 = \{Eno, En, Es\}$ 和 $R_3 = \{Es, Eg\}$

第五章 SQL

1. SQL 特点

①综合统一②高度非过程化③面向集合的操作方法④一种语法结构, 两种使用方式(自含式语言、嵌入式语言)⑤语言简洁、语法简单、易学易用

2. 基本表、查询表和视图表的特点和区别

(1) 基本表: 实际存在的表。即, 基本表是实表。DBMS 保存并维护基本表的数据和元数据。在一个基本表上, 可以创建并维护若干个索引表。

(2) 查询表: 用于存放查询的中间结果, 以及查询结果的表。查询表是临时表。查询表的数据和元数据都是“临时”的。

(3) 视图表: 简称: 视图。它是由基本表或其它视图表“导出”的表。视图表是“虚”表。视图本身没有独立存在的数据, 没有实际的物理记录, 逻辑存在, 不占用物理空间。

作用: 从用户的角度隐藏数据, 简化查询书写方式(保护数据, 简化查询)

3. SQL 中的数据类型

char, varchar, int, smallint, numeric(存任意精度), real(double precision), float, NULL, date, time, timestamp

interval [时间间隔数量][时间间隔单位]

4. 表创建、修改和删除 见 PPT 63-72
5. 索引的创建和删除 见 PPT 73、74
6. SQL 查询语句结构 如右
7. 空值概念

一个典型的SQL查询具有如下形式:

```
select A1, A2, ..., An
from r1, r2, ..., rm
where P1
[Group by Ai [having P2]]
[Order by Aj ASC|DESC,.....]
Ais represent attributes
rjs represent relations
P1, P2 are predicate (条件)
```

Select...From...Where等同于下面的关系代数表达式:

$\prod_{A_1, A_2, \dots, A_n} (\sigma_{P_1} (r_1 \times r_2 \times \dots \times r_m))$
SQL 查询的结果是包 (Bag)。

(1) 概念: SQL 中用“null”表示信息缺失或不确定。用“is null”进行空值测试

(2) 运算: ①与 NULL 相关的数学运算, 结果都是 NULL ②NULL 与任意值比较返回 unknown

③聚集函数会忽略 NULL

(3) unknown: 布尔值的一种, 在 where 引出的条件子句中, 如果条件返回的是 unknown, 则认为是 false

- OR: (unknown or true) = true, (unknown or false) = unknown, (unknown or unknown) = unknown
- AND: (true and unknown) = unknown, (false and unknown) = false, (unknown and unknown) = unknown
- NOT: (not unknown) = unknown

8. 聚集函数 (sum, count, avg, max, min) 见 PPT 30-36

9. 基本表的查询 见 PPT 10-23 字符串运算 (★选择题★) PPT 24、25 ORDER PPT26 集合运算 (并) PPT27-29

10. 嵌套子查询 见 PPT40-56 连接运算 (忽略外连接) PPT 57-61

第六章 SQL 约束

1. 键码 (主码) 约束 PK

PRIMARY KEY(属性 (组)), 对应了实体的完整性约束

2. 单值约束 UNIQUE

在某条属性声明时写 UNIQUE, 可以取 NULL 值

• PK 与 UNIQUE 的区别:

①一个表中 PK 只能出现一次, UNIQUE 可以出现多次: DBMS 在 PK 上建立聚族索引, 而聚族索引的顺序与表中记录的存贮顺序一致。但在 Unique 上建立的是唯一索引, 其与存贮顺序无关

②UNIQUE 可以取 NULL 值, PK 不可以

• PK 和 UNIQUE 的作用:

当对表进行插入、修改操作时, DBMS 将进行主码约束或单值约束检查

3. 取值约束 CHECK (全局&非全局)

(1) 加在某属性末尾: 对该属性取值约束

(2) 全局约束 (元组间约束) 如右图

4. 参照完整性约束

若关系 R2 对 R1 有 REFERENCES, 那么 R2 中的任意元组在外码 FK 上的取值, 必须是 R1 上该码的取值, 不能有 R1 该码没取过的值, 或者取 NULL

• 保持参照完整性的三个可选策略

①任何破坏该性质的操作, 都会被 DBMS 拒绝

②级联策略: 修改被参照的关系元组时, 系统级联修改外码所在元组或外码属性值

③置空策略: 修改被参照的关系元组时, 系统修改外码属性值为 NULL

5. 断言

指数据库中数据要满足的条件。前面讲的各种约束都是断言的特殊形式。但断言的功能更强大, 可以定义各种约束。create assertion <断言名>check(条件)

6. 触发器

触发器是一个语句集合 (规则), 当数据库作修改时, 被系统自动执行。需要指定执行条件时机&采取动作。CREATE TRIGGER <NAME> BEFORE/AFTER <时机> [OF 属性] ON <表> [FOR EACH ROW] WHEN(...) [相关操作]

第七章 数据库的安全与控制

1. 数据库安全性概念

(1) 概念: 保护数据库, 防止因用户非法使用数据库造成数据泄露、更改或破坏

(2) 一般方法: ①用户身份标识和鉴定②存取权限控制③定义并使用视图④审计制度⑤数据加密、

(3) 存取权限基本类型: ①读取②插入③更新④删除

```
E.g. CREATE TABLE Persons
( Name CHAR(30),
  Title CHAR(10) DEFAULT 'Dr.',
  Gender CHAR(1),
  Birthdate DATETIME,
  PRIMARY KEY (name),
  CHECK (( NOT(Title = 'Dr.%')
          OR ( Birthdate < '01/01/1981' ) ) ) )
```

注: 年龄足够大的, 才能用“Dr.”头衔。

(4) SQL 授权/收回

①授权: **GRANT <权限> ON <关系名或视图名> TO <用户> [WITH GRANT OPTION]**

注: <权限>例如 SELECT,INSERT; 最后面的[WITH...]仅限 owner 或 DBA 使用, 由此获得“授权”权限的<用户>可以将获得的权限继续传递给其他用户。GRANT OPTION 也属于权限, 可以被 owner 或 DBA 收回

②收回: **REVOKE <权限> ON <关系名或视图名> FROM <用户表> [CASCADE]**

注: CASCADE 是级联删除选项; 如果要收回某用户授予 SELECT 的权限, 需要在<权限>填写 GRANT OPTION FOR SELECT, 注意 FOR 的使用

③局限性: SQL 不支持元组级别的权限控制, 比如我们无法让 A 学生在成绩单中只看到 A 学生所在元组

2. 事务管理

(1) 概念: 事务是**访问**并可能**更新**数据库**数据**的一个**程序执行单位**

(2) 性质 ACID

①原子性 Atomicity: 组成事务的**操作不可分割**

②一致性 Consistency: 一个事务必须是一个**正确程序**, 使数据库从一个**一致状态**转为**另一个一致状态**

③隔离性 Isolation: 一个正在执行的事务在**提交前**, **不允许对共享数据所做的改变提交给其他事务用**

④持久性 Durability: 一旦事务**提交**, 即使**系统发生故障**, **也不能丢失**该事务的**执行结果**

3. 事务的并发操作

(1) 并发: 多个事务**同时存取相同**的数据

(2) 带来的问题: **数据一致性问题**①丢失修改②读入脏数据③不可重复的读

(3) 封锁 ①排它锁 X: 写锁 ②共享锁 S: 读锁 仅 S 与 S 之间可以相容

(4) 三级锁协议

①一级锁协议: 事务在**修改数据之前**, 必须先对其加 **X 锁**, 直到事务**结束/提交/回退后**, 才**释放**, 解决“**丢失修改**”

②二级锁协议: 在**一级锁的基础上**, 事务在**读取数据之前**, 必须先对其加 **S 锁**, **读入后立即释放**, 解决“**读脏数据**”

③三级锁协议: 在**一级锁的基础上**, 事务在**读数据之前**, 必先对其加 S 锁, 直到事务**结束/提交/回退后**, 才**释放**, 解决“**不可重复读**”

(5) 并发调度的可串行性

如果多个事务的**并发**调度执行过程的**结果**, 与它们的一个**串行**执行**过程产生的结果相同**, 则称并发调度是**可串行化**的。

(6) 两阶段锁协议

每个事务分两个阶段提出锁操作①获得锁阶段: 任何事务可以申请获得锁, 但不能释放锁②释放锁阶段: 任何事务可以申请释放锁, 但不能获得新锁。如果**所有的事务都遵守“两段锁协议”**, 则**这些事务是可串行化的**。