



数据库期末复习

1. SQL 查询

原则：先选择投影，再进行连接

$$\pi_{Sno, Sname}(Cno = 'C2' (S \bowtie SC))$$
$$\pi_{Sno, Sname}(S) \bowtie (\pi_{Sno}(Cno = 'C2' (SC)))$$

2. Check point:

① 选修 C2 或 C4 的学号

$$\pi_{Sno}((Cno = 'C2' \vee Cno = 'C4') (SC \times SC))$$

② C2 + C4

$$\pi_1(S1=4 \wedge S2='C2' \wedge S5='C4') (SC \times SC))$$

③ 不学 C2 课的学号 (减法思维)

$$\pi_{Sno}(S) - \pi_{Sno}(Cno = 'C2' (SC))$$

3. SQL 特点

(1) 综合统一
集 DDL, DML, DCL 于一体。

(2) 高度非过程化
不用考虑如何实现，只需提出“做什么”，不关心“怎么做”。

(3) 面向集合的操作方式

查询、插入、删除、修改的操作对象及结果都是集合。

(4) 以同一种语法结构提供两种使用方法

可交互式和嵌入式使用。

(5) 以简捷的自然语言作为操作语言

定义了少量的关键字实现对数据库的定义、操纵和控制功能。



4. 完整性约束

• 创建基本表常用的完整性约束

(1) 主码约束：PRIMARY KEY

(2) 唯一性约束：UNIQUE(不能取相同值但允许多个空值)

(3) 非空值约束：NOT NULL

(4) 参照完整性约束：

FOREIGN KEY (<列名>) REFERENCES <表名>(<列名>)

(5) CHECK约束：CHECK (<谓词>)

(6) 断言(Assertion)约束（自学）

```

① CREATE TABLE SC(
  Sno CHAR(5),
  Cno CHAR(3),
  Grade INT CHECK (Grade >= 0 AND Grade <= 100),
  PRIMARY KEY (Sno, Cno),
  FOREIGN KEY (Sno) REFERENCES S(Sno),
  FOREIGN KEY (Cno) REFERENCES C(Cno)
);

```

② DROP TABLE Student;
↓
表名

ALTER TABLE Student ADD Senroll DATE NOT NULL;
表名 新列名 类型 可加约束

ALTER TABLE Student ALTER Sage SMALLINT;
列名

ALTER TABLE Student DROP UNIQUE(Sname);
已有约束

ALTER TABLE Student drop Sroll:
某列

- ③ INSERT INTO Student VALUES ('95020', '男');
INSERT INTO SC(Sno, Cno) VALUES ('95020', '1');
从左至右码及非空值

子查询结果

INSERT INTO Deptage (Sdept, Avgage)

{ SELECT Sdept, AVG(Sage)
FROM Student
GROUP BY Sdept;

- ④ UPDATE Student SET Sage = 22;
WHERE Sno = '95001';

UPDATE Student
SET Sage = Sage + 1;

正确的语句：

UPDATE SC
SET Grade = 0
WHERE 'CS' → 必须在前面
{ SELECT Sdept
FROM Student
WHERE Student.Sno = SC.Sno };

DELETE FROM Student
WHERE Sno = '95019';

DELETE
FROM SC
WHERE Sno IN (SELECT Sno
FROM Student
WHERE Student.Sdept = 'CS');
或
DELETE
FROM SC
WHERE 'CS' =
(SELECT Sdept
FROM Student
WHERE Student.Sno = SC.Sno);

- ⑤ SELECT [ALL | DISTINCT] <目标列表达式>
[, <目标列表达式>] ...
FROM <表名或视图名1> [, <表名或视图名2>] ...
[WHERE <条件表达式>]
[GROUP BY <列名> [HAVING <条件表达式>]]
[ORDER BY <列名> [ASC | DESC]]; → 缺省表升序

相关子句的说明：

- SELECT子句：指定要显示的属性
- FROM子句：指定查询的数据（基本表或视图）
- WHERE子句：指定查询条件
- GROUP BY子句：对查询结果按指定列分组，列出相同的记录为一组，通常再在结果上施加函数或算术表达式
- HAVING短语：筛选出所有满足指定条件的组
- ORDER BY子句：对查询结果按指定列值升序或降序排序

SELECT Sname NAME, 1996-Sage BIRTHYEAR
From Student; 列别名

★ SQL中，不等于：<>, !=

★ 在..之间 WHERE Sage BETWEEN 20 AND 30;

WHERE Sdept NOT IN ('IS', 'MA', 'CS');

⑥ 字符串匹配 → 转义通配符

[NOT] LIKE '<匹配模板>' [ESCAPE '<换码字符>']

* 匹配模板：固定字符串或含通配符的字符串。

%：任意长度字符串

a%b：acb, aabgb, ab

-：任意单个字符

a-b：acb, cfb

» 用转义符将通配符转义为普通字符

[例19] 查询DB_Design课程的课程号和学分。

SELECT Cno, Ccredit

FROM Course

WHERE Cname LIKE 'DB_Design' ESCAPE '\';

转义符`表示模板中出现在其后的第一个字符不再是通配符，而是字符本身。

⑨ 聚合查询

子查询不能使用 ORDER BY 子句

SELECT S1.Sno, S1.Sname, S1.Sdept

FROM Student S1, Student S2 → 别名

WHERE S1.Sdept = S2.Sdept AND

S2.Sname = '刘晨';

⑩ 聚集函数

COUNT, SUM, AVG, MAX, MIN

SELECT COUNT(DISTINCT Sno)

FROM SC;

注：用DISTINCT以避免重复计算学生人数

HAVING子句作用于且只能作用于各组之下。

GROUP BY子句的作用对象是查询的中间结果表

【对查询结果分组，函数将作用于整个数据结果】

【对查询结果分组后，函数将分组作用于每组】

【用GROUP BY子句后，SELECT子句的别名表中只能出现分组属性和聚集函数】

[例31] 查询有3门以上课程是90分以上的学号及其(90分以上的)课程数。

SELECT Sno, COUNT(*)
FROM SC
WHERE Grade >= 90
GROUP BY Sno
HAVING COUNT(*) >= 3;

⑪ ANY, ALL

[例30] 查询其他系中信息系某些学年的小的学生姓名和年龄。

SELECT Sname, Sage

FROM Student

WHERE Sdept != ANY (SELECT Sdept

FROM Student

WHERE Sdept = 'IS')

AND Sdept <> 'IS'; //这是又费掉中的条件

5. 视图 View

视图对应三级模式/两级映射中体系结构中的外模式和外模式/模式的映象。

建立视图：

视图定义 DBMS → 数据字典 ← 视图查询

CREATE VIEW IS_Student

AS

SELECT Sno, Sname, Sage

FROM Student

WHERE Sdept = 'IS' ;

CREATE VIEW S_G(Sno, Gavg)

AS

SELECT Sno, AVG(Grade)

FROM SC

GROUP BY Sno;

SELECT

Data

指名列

⑦ 外连接

SELECT [ALL | DISTINCT] <目标列表达式1>

, <目标列表达式2>] ...

FROM <table_source1> INNER|LEFT | RIGHT | FULL

[OUTER] JOIN < table_source2 > ON <search_condition>

[WHERE <条件表达式>]

SELECT Student.Sno, Sname, Sex, Sage, Sdept, Cno, Grade

FROM Student LEFT OUTER JOIN SC

ON Student.Sno = SC.Sno;

⑧ 相关子查询与不相关子查询

此类查询称为相关子查询，即子查询的条件与父查询当前值相关。

EXISTS 不返回数据 只产生 True or False

SELECT Sname

FROM Student

WHERE EXISTS

(SELECT *

FROM SC

WHERE Sno = Student.Sno AND Cno = 'c1') :

★ 关系代数中

f: 重命名运算 = As

Example: f_C(CC)

不能作用在 WHERE 子句中

用 HAVING 实现

f

→ HAVING 实现

f

→ WHERE 子句

f

→ HAVING 实现

f

→ WHERE 子句

f

6. 连接之前对选择的数据首先做投影

$\pi_{Sno, Pno} (\sigma_{Pno = 'J1'} (SP)) \bowtie \pi_{Pno} (\sigma_{Pno = 'E2'} (P))$

关系理论:

主属性: 非码中出现的属性。

候选码: 关系 R(U) 中, 能够决定所有属性的属性或属性组。K_R

关系的完整性

① 实体完整性: 主码非空且唯一

Sno CHAR(9) PRIMARY KEY,
列级
PRIMARY KEY (Sno, Cno) 表级
PRIMARY KEY (Sno)

② 参照完整性:

$R(K_r, F, \dots)$

参照关系

K_r, K_s : 主码, F : R 的外码

K_s, F 定义在同一个域上

F : 一个一组合属性, 要求:

{ 或者取空值 (F 的每个属性值均为空值) → 但 F 不能为 R 的主属性
或者等于 S 中某个元组的主码值。 }

FOREIGN KEY (Sno) REFERENCES $S(Sno)$,
表级 Sno 参照 S 中的 Sno

Sno CHAR(5) REFERENCES $S(Sno)$,
列级

③ 用户定义完整性

非空 NOT NULL

唯一 UNIQUE

约束 CHECK

元组上 (局部属性)

单个属性

CREATE TABLE Student(
 Sno CHAR(9) PRIMARY KEY,
 $Sname$ CHAR(8) NOT NULL UNIQUE,
 $Ssex$ CHAR(2) CHECK ($Ssex$ IN ('男', '女'))

④ 完整性约束命名

```
CREATE TABLE Student
  ( $Sno$  NUMERIC(6)
   CONSTRAINT C1 CHECK ( $Sno$  BETWEEN 90000 AND
   99999),
    $Sname$  CHAR(20)
   CONSTRAINT C2 NOT NULL,
    $Sage$  NUMERIC(3)
   CONSTRAINT C3 CHECK ( $Sage$  < 30),
    $Ssex$  CHAR(2)
   CONSTRAINT C4 CHECK ( $Ssex$  IN ('男', '女')),
   CONSTRAINT StudentKey PRIMARY KEY( $Sno$ )
);
```

思路: 先删除原来的约束条件, 再增加新的约束条件

ALTER TABLE Student DROP CONSTRAINT C1;

ALTER TABLE Student

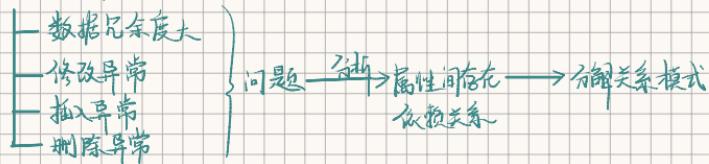
ADD CONSTRAINT C1 CHECK (Sno BETWEEN 900000 AND 999999);

ALTER TABLE Student DROP CONSTRAINT C3;

ALTER TABLE Student

ADD CONSTRAINT C3 CHECK ($Sage$ < 40);

关系模式规范化



函数依赖 \longrightarrow 语义清晰 $R(u, X \subseteq u, Y \subseteq u)$

$$X \rightarrow Y \quad X \text{ 函数确定 } Y \quad | \quad Y \text{ 函数依赖于 } X$$

\downarrow 关系模式 R 的所有关系实例均满足

- ① 若 $X \rightarrow Y$, 但 $Y \not\subseteq X$, 则称 $X \rightarrow Y$ 是非平凡的函数依赖
- ② 若 $X \rightarrow Y$, 但 $Y \subseteq X$, 则称 $X \rightarrow Y$ 是平凡的函数依赖 (Trivial FD)
- ③ 若 $X \rightarrow Y$, 并且 $Y \rightarrow X$, 则记为 $X \xleftarrow{\Delta} Y$. (X 与 Y 相互决定)
- ④ 若 Y 不函数依赖于 X , 则记为 $X \not\rightarrow Y$.

完全函数依赖

$X \rightarrow Y$, 且 $\forall X' \neq X, X' \rightarrow Y$, 则 $X \xrightarrow{F} Y$

部分函数依赖

$X \rightarrow Y$, 且 $\exists X' \neq X, X' \rightarrow Y$ 则 $X \xrightarrow{P} Y$

传递函数依赖

$X \rightarrow Y, Y \neq X, Y \rightarrow X, Y \rightarrow Z$ 有 $X \rightarrow Z$ 则 $X \xrightarrow{T} Z$

\square 保证不转化为

若 $Y \subseteq X$ 则 $Y \rightarrow X$ 直接依赖

$\exists X' = Y \neq X, X \rightarrow Z$, 则 $X \xrightarrow{P} Z$

候选码

$R(u, F), k \subseteq u$, if $k \xrightarrow{F} u$, 则 k 为 R 的候选码

主码: 若干候选码中的一个

外码: $R(u, F), X \subseteq u$, X 不是 R 的码而是 S 的码
 $S(u', F')$,

范式判别

1NF: R 中的所有属性都是不可分的基本数据项
 $R \in 1NF$

2NF: $R \in 1NF$, 非主属性完全函数依赖于 R 的码
 $R \in 2NF$ *

3NF: $R<u, F>$ 中码 X , 属性组 Y , 非主属性 Z ($Z \notin Y$)

不存在 $X \rightarrow Y$, $Y \rightarrow Z$ 则 $R \in 3NF$

不存在非主属性对码的传递依赖 (只有直接依赖)
★

BCNF: $R<u, F> \in 1NF$, $\forall X \rightarrow Y$ ($Y \neq X$), X 必包含码

$R \in BCNF$

★每一个函数依赖的决定因素都包含码

例: 关系模式 $SJP(S, J, P)$ 中, S 是学生, J 表示课程, P 表示名次。每一学生选修每门课程的成绩都有一定名次, 且名次不重复
★

FD: $(S, J) \rightarrow P$, $(J, P) \rightarrow S$

码: (S, J) , (J, P)

非主属性: 无

①不存在非主属性对码的部分依赖 $SJP \in 2NF$

②不存在非主属性对码的传递依赖 $SJP \in 3NF$

③每一个函数依赖的决定因素都包含码 $SJP \in BCNF$

结论: BCNF 消除了主属性对码的部分依赖和传递依赖。
在函数依赖的范畴内解决了数据插入、异常和删除异常。
但可能存在数据冗余和修改复杂。

Armstrong 公理系统 | 函数依赖公理系统

$F^+ = X \rightarrow Y$ F 逻辑蕴含 $X \rightarrow Y$

$R<u, F>$, $\forall r \in R$, if $X \rightarrow Y$ then $F^+ = X \rightarrow Y$

①自反律: $Y \subseteq X \subseteq u$, 则 $F^+ = X \rightarrow Y$

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

$$(A, B) \rightarrow A, (A, B) \rightarrow B, (A, B) \rightarrow (A, B) \in F^+$$

②增广律 $F^+ = X \rightarrow Y$, $Z \subseteq U$ then $F^+ = XZ \rightarrow YZ$

$$U = \{A, B, C, D\}, F = \{A \rightarrow B\}$$

$$\overbrace{XZ}^{=} = X \cup Z$$

$$AC \rightarrow BC \subset F^+$$

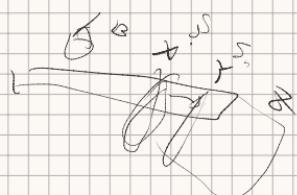
③传递律 $F^+ = X \rightarrow Y$, $F^+ = Y \rightarrow Z$ then $F^+ = X \rightarrow Z$

$$U = \{A, B, C, D\}, F = \{A \rightarrow B, B \rightarrow C\}$$

$$\downarrow$$

$$A \rightarrow C \subset F$$

④合并不规则 $\{X \rightarrow Y, X \rightarrow Z\} = X \rightarrow YZ$



⑤ 分解规则 $\{X \rightarrow Y, Z \subseteq Y\} = X \rightarrow Z$

⑥ 传递递 $\{X \rightarrow Y, Y \rightarrow Z\} = X \rightarrow Z$



F的闭包 F⁺ 函数依赖集闭包

$R \subseteq U, F \supseteq$ 中 F 所逻辑蕴含的函数依赖的全体.

■ F^+ 的意义：包含了给定函数依赖集 F(部分)所蕴涵的属性集 U 上的全部函数依赖。

属性集的闭包

$R \subseteq U, F \supseteq$ 且 $X \subseteq U$, 则 X 关于函数依赖集 F 下的闭包:

$$X_F^+ = \{A \mid F \vdash A\}$$

■ 求 X_F^+ 的算法：

(1) 令 $X^{(0)} = X$, $i = 0$;

(2) 令 $X^{(i+1)} = X^{(i)} U \{A \mid (\exists V)(\exists W)(V \rightarrow W \in F \wedge V \subseteq X^{(i)} \wedge A = W)\}$;

(3) 若已没有 $V \rightarrow W \in F$, 使得 $X^{(i+1)} = X^{(i)}$, 算法结束, $X_F^+ = X^{(i)}$; 否则, 令 $i = i + 1$, 转(2)。

例：已知关系模式 $R \subseteq U, F$, 其中 $U = \{A, B, C, D, E\}$,

$F = \{AB \rightarrow C, B \rightarrow D, C \rightarrow E, EC \rightarrow B, AC \rightarrow B\}$, 求 $(AB)_F^+$.

$$\text{解: } (AB)_F^+ = \{ABCDE\}$$

(1) $X \subseteq X_F^+$:

(2) 逐一考察 F 中的 FD, 如果存在 $V \rightarrow W \in F$, 且 V 中的全部属性都出现在当前的 X_F^+ 中, 将属性集 W 并入 X_F^+ ;

(3) 当当前的 X_F^+ 包含全部属性 U, 或者按(2)重新遍历 F 而没有对当前 X_F^+ 增加任何属性时算法结束。

判断函数依赖是否成立的方法

■ 定理：设 F 为属性集 U 上的一组函数依赖， $X, Y \subseteq U$, $X \rightarrow Y$ 能由 F 根据 Armstrong 公理导出的充分必要条件是 $Y \subseteq X_F^+$.



■ 最小函数依赖集求解方法：

(1) 对 F 中每一函数依赖 $X \rightarrow Y$, 若 $Y = A_1 A_2 \dots A_m (m \geq 2)$, 则用

$[X \rightarrow A_i | i=1, \dots, m]$ 替换 $X \rightarrow Y$; (等价变换) 右部冗属性

(2) 对 F 中每一函数依赖 $X \rightarrow A$, 若 $X = B_1 B_2 \dots B_n$, 逐一考察 B_i , 若 B_i 左部去冗余 $A \in (X - B_i)^+$, 则用 $(X - B_i) - A$ 替换 $X \rightarrow A$, B_i 称为冗余属性; (等价变换)

(3) 对于 F 中的每一函数依赖 $X \rightarrow A$, 若 $A \in X^*_{F \setminus \{X \rightarrow A\}}$, 则从 F 中去掉 $X \rightarrow A$. (等价变换)

去掉传递可得的 FD

例：设关系模式 $R(ABCDE)$ 上的函数依赖集 $F = \{A \rightarrow BC, BCD \rightarrow E, B \rightarrow D, A \rightarrow D, E \rightarrow A\}$,

$B \rightarrow D, A \rightarrow D, E \rightarrow A$, 求 F 的最小函数依赖集。

解：① 对每个函数依赖右部属性分离，得：

$$F_1 = [A \rightarrow B, A \rightarrow C, BCD \rightarrow E, B \rightarrow D, A \rightarrow D, E \rightarrow A]$$

② 去掉左部冗属性

$\because (BCD)^+ = BCDEA$, 包含 E, $BCD \rightarrow E$ 中的 D 为冗余属性, 以 $BC \rightarrow E$ 取代 $BCD \rightarrow E$, 得：

$$F_2 = [A \rightarrow B, A \rightarrow C, BC \rightarrow E, B \rightarrow D, A \rightarrow D, E \rightarrow A]$$

③ 去掉多冗函数依赖

$\therefore A \rightarrow B, B \rightarrow D$ 可以得到 $A \rightarrow D$. 故 $A \rightarrow D$ 多余, 去掉, 得 $F_{min} = [A \rightarrow B, A \rightarrow C, BC \rightarrow E, B \rightarrow D, E \rightarrow A]$

候选码求解算法

■ 候选码求解算法

◆ 分析:

对于给定的关系模式 $R \leftarrow U, F \rightarrow$, 依照函数依赖集F将U中的属性分为以下四类:

L类属性: 在F中只出现在函数依赖的左部的属性;

R类属性: 在F中只出现在函数依赖的右部的属性;

LR类属性: 分别出现在F中的函数依赖左部和右部的属性;

N类属性: 不在F中的函数依赖中出现的属性。

◆ 结论:

① L类属性和N类属性必包含于任何候选码中;

② R类属性必包含于任何候选码中;

③ LR类属性不能确定是否在候选码中。

例: 设 $R \leftarrow U, F \rightarrow$, 其中:

$U = \{A, B, C, D, E\}$, $F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$, 求R的
所有候选码。

解: (1) R中无L、N类属性,
A, B, C, D, E均为LR类属性,
故 $X = \Phi$, $Y = \{A, B, C, D, E\}$;

(2) $X_F = \Phi \neq U$ (但不
能说明R的候选码不唯一?)

(3) 取A, 则

$A_F = ABCDE = U$, A为候选码;

取B、C、D, 其闭包均不
等于全属性集U;

取E, 则 $E_F = ABCDE = U$,

E为候选码;

$Y = \{B, C, D\}$

算法:

对于给定的关系模式 $R \leftarrow U$,

$F \rightarrow$, 其中U为属性集合, F为函数依赖集。

(1) 依照函数依赖集F将R中的所有属性分为L类、R类、LR类和N类属性。令 $X \leftarrow \{ \}$ 、 $Y \leftarrow \{ \}$ 为属性的集合, $Z \leftarrow \{ \}$ 为LR类属性集合。

(2) 若 $X_F = U$, 则X为R的唯一候选码(‘), 结束; 否则, 转(3);

(3) 选取Y中的单一属性Y₁, 若 $(X \cup Y_1)_F = U$, 则 XY_1 为候选码(‘), 否则 $Y = Y - Y_1$ 转(4);

(4) 依次取Y中的任意两个、三个……属性与XZ组成属性组, 若 $(X \cup Y_1 \cup Y_2)_F = U$, 求其关于Y的闭包 $(Y_2)_F$, 若 $(Y_2)_F = U$, 则 XYZ 为候选码, 直到满足Y中的所有限制为止。算法结束。

(5) BCD包含已求得的候选码

BC, BCD 不是码, 结束。

故关系R的候选码为: A, E, BC, CD。

例: 设 $R \leftarrow U, F \rightarrow$, 其中:

$U = \{A, B, C, D, E\}$, $F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$, 求R的
所有候选码。

(3) 取A, 则

$A_F = ABCDE = U$, A为候选码;

取B、C、D, 其闭包均不
等于全属性集U;

取E, 则 $E_F = ABCDE = U$,

E为候选码;

$Y = \{B, C, D\}$

(4) 在Y中任取两个属性判定

$(BC)_F = BCDEA = U$, BC
为候选码;

$(BD)_F$ 不等于全属性集U,
 BD 不是候选码;

$(CD)_F = CDEAB = U$, CD
为候选码;

模式分解

$R \leftarrow U, F \rightarrow$ 一个分解 $p = \{R_1 \leftarrow U_1, F_1 \rightarrow, R_2 \leftarrow U_2, F_2 \rightarrow, \dots, R_n \leftarrow U_n, F_n \rightarrow\}$

$U = \bigcup_{i=1}^n U_i$, $U_i \cap U_j = \emptyset$, $1 \leq i, j \leq n$, $i \neq j$

\downarrow
U_i对U的划分

$F_i \iff \{x \rightarrow y \mid x \in F \wedge y \in U_i\}$

模式分解的三条准则

◆ 分解具有无损连接性

分解后的后关系通过自然连接可以“恢复”原始的关系

◆ 分解应保持函数依赖

函数依赖是属性间自身具有的性质, 通常的查询与此有关, 故应保持。

◆ 分解既要保持函数依赖, 又要具有无损连接性

判定无损连接性

无损连接性的判定算法——a. 构表法

设 $p = \{R_1 \leftarrow U_1, F_1 \rightarrow, R_2 \leftarrow U_2, F_2 \rightarrow, \dots, R_n \leftarrow U_n, F_n \rightarrow\}$ 是

关系模式 $R \leftarrow U, F \rightarrow$ 的一个分解,

$U = \{A_1, \dots, A_n\}$, $F = \{A_1 \rightarrow B_1, \dots, A_n \rightarrow B_n\}$

(B_i 为最小数据粒度, 即不是 A_i , 可提高求解效率), 并设 FD_i 为 $X_i \rightarrow A_i$,

$\rightarrow A_i$ 。

[1] 建立一个 $n \times k$ 行的表,

每一列对应一个属性, 每一

行对应一个关系模式, 若属

性A_i属于U_j, 则在j行交叉

处填上a_{ij}, 否则填上b_{ij}

例: 已知 $R \leftarrow U, F \rightarrow$, $U = \{A, B, C, D, E\}$,

$F = AB \rightarrow C, D \rightarrow E, C \rightarrow D, B \rightarrow A$

$R = \{R_1 \leftarrow U_1, F_1 \rightarrow, R_2 \leftarrow U_2, F_2 \rightarrow, \dots, R_n \leftarrow U_n, F_n \rightarrow\}$

(R_1, R_2, \dots, R_n 为最小数据粒度, 即不是 A, B, C, D, E), 并且分解p是否为无损连接的分解。

解: (1) 构造初始表:

$R_1 \leftarrow U$	A	B	C	D	E
(ABC)	a ₁₁	a ₁₂	b ₁₃	b ₁₄	b ₁₅
(CD)	b ₂₁	b ₂₂	a ₂₃	a ₂₄	b ₂₅
(DE)	b ₃₁	b ₃₂	b ₃₃	a ₃₄	a ₃₅

(2) 如果在一 σ -Chase中的某次

更改之后出现形如a₁₁, a₁₂, ..., a_{1n}的

行, 算法结束。分解p具有无

损连接性。否则, 继续下一 σ -

Chase过程。直到在一 σ -Chase之

后表无任何变化时算作结束(此

时未出现形如a₁₁, a₁₂, ..., a_{1n}的行)。

分解p不具有无损连接性。

(2) 把表看作R的一个关系, 依

次检查每一FD_i在表中是否成立,

若不成立, 做如下修改使其成

立: 找到X_i所对应的列中具有相

同符号的那些行, 如果A_i列出现

a_{ij}, 将这些行的A_i列换成a_{ij},

否则全部改为b_{ij}, m是行号的

最小值: (称为一 σ -Chase过程)

(3) σ : $\dots, (F = AB \rightarrow C,$

$D \rightarrow E, C \rightarrow D), \dots$

解: (1) 构造初始表:

$R_1 \leftarrow U$	A	B	C	D	E
(ABC)	a ₁₁	a ₁₂	a ₁₃	b ₁₄	b ₁₅
(CD)	b ₂₁	b ₂₂	a ₂₃	a ₂₄	b ₂₅
(DE)	b ₃₁	b ₃₂	b ₃₃	a ₃₄	a ₃₅

(2) 第一 σ -Chase:

① $AB \rightarrow C$: 无AB列上取值

相同的行, 不做修改;

② $D \rightarrow E$: 第二、三行在D

上的取值均为a₃₄, 将这两行在E上的取

值修改为a₃₅;

(3) 第二 σ -Chase:

① $AB \rightarrow C$: 不改变;

② $D \rightarrow E$: 第一、二、三行在

D上的取值为a₁₄, 将它们在E上

的值改写为a₁₅;

(4) 表第一行出现a₁₁, a₁₂, a₁₃, a₁₄, a₁₅, 放弃解p无损分解。

(例): ($F = AB \rightarrow C,$

$D \rightarrow E, C \rightarrow D$),

(2) 第一 σ -Chase:

①

②

$R_1 \leftarrow U$	A	B	C	D	E
(ABC)	a ₁₁	a ₁₂	a ₁₃	b ₁₄	b ₁₅
(CD)	b ₂₁	b ₂₂	a ₂₃	a ₂₄	b ₂₅
(DE)	b ₃₁	b ₃₂	b ₃₃	a ₃₄	a ₃₅

③ $C \rightarrow D$: 第一、二行在C
上的取值为a₁₃, 将这两行在D
上的值改写为a₁₄;

无损连接性——b. 判断定理

定理：关系模式 $R < U, F >$ 的一个分解 $\rho = \{ R_1 < U_1, F_1 >, R_2 < U_2, F_2 > \}$ 具有无损连接性的充分必要条件是：

$$U_1 \cap U_2 \rightarrow U_1 - U_2 \in F^* \text{ 或 } U_1 \cap U_2 \rightarrow U_2 - U_1 \in F^*$$

注：① 单独的一个条件是充分条件而非必要条件；

② 此定理适用于一分为二的模式分解无损连接性的判定。

例：学生关系 S ($Sno, Sname, Sex, Dept, DeptManager$) 分解为 $S1(Sno, Sname, Sex, Dept)$ 和 $S2(Dept, DeptManager)$, $DFS = Dept, D - S = DeptManager, Dept \rightarrow DeptManager$ 为原关系中的函数依赖，此分解为无损连接的分解。

$$R(A, B, C, D, E), F = \{A \rightarrow B, BC \rightarrow D, DE \rightarrow A\}$$

解：
 $L = \{C, E\}, N = \emptyset$
 $R = \emptyset, LR = \{A, B, D\}$

① 经分析得： $X = L \cup N = \{C, E\}, Y = LR = \{A, B, D\}$

② 求 X 关于 F 的闭包：

$$X_F^+ = \{C, E\} \neq U$$

③ 取 Y 中的 A ，得： $(XA)_F^+ = \{A, B, C, D, E\} = U$

故 (ACE) 为 X 的候选码

取 Y 中的 B ，得： $(XB)_F^+ = \{A, B, C, D, E\} = U$

取 Y 中的 D ，得： $(XD)_F^+ = \{A, B, C, D, E\} = U$

故 $(BCF), (DCE)$ 为 X 的候选码

④ 全 $Y = Y - \{A, B, D\} = \emptyset$

故 X 的所有码为 $(ACE), (BCE), (CDE)$

数据库设计

概念结构设计 —— ER 模型

■ 确定实体与属性的两条准则：

- (1) 属性是不可再分的数据项，属性不可以再有属性；
- (2) 属性不能与其他实体发生联系，联系只能存在~~于实体与实体之间~~。

■ 作实体还是作联系：

联系通常作为对事件的描述，发生于多个实体之间。

■ 联系的维度：

维度描述了参与联系的实体的个数，应保证~~事件（联系）~~参与者的完整性、刻画的精密度及应用需求。

逻辑结构设计

关系模式的转换

① 实体的转换

● 实体及其普遍属性 (非 派生属性、组合属性、多值属性)

构成一个关系模式：

- 实体名及关系名
- 实体的普遍属性转换为关系的属性
- 实体标识符转换为关系的码

● 派生属性不转换 (丢弃)：

- 组合属性丢弃，而只取其分量；

● 每个多值属性分别与实体标识符另构成新的关系模式。

转换为如下关系模式：

学生(学号, 性别, 年龄, 出生日期)

姓名(学号, 姓名)

② 1:n 联系的转换

① 转换为一个独立的关系模式

- 关系的属性：与该联系相关的各实体的码以及联系本身的属性
- 关系的码：n端实体的码



转换成如下关系模式：

班级(班号, 班级, 班主任, 入学时间)

学生(学号, 姓名, 性别, 年龄)

成员(学生, 班号)

② 与n端对应的关系模式合并(常用)

- 合并后关系的属性：在n端关系中加入1端关系的码和联系本身的属性
- 合并后关系的码：不变



转换成如下关系模式：

班级(班号, 班级, 班主任, 入学时间)

学生(学号, 姓名, 性别, 年龄)

注：实际使用时建议采取这种方式以减少冗余数据。因为多个一个关系模式的数据修改过程需要进行逻辑运算，而两个关系的更新则相对简单。

③ 三个或三个以上实体间的一个多元联系转换为一个关系模式，也可以分解为多个二元联系。

④ 转换为一个独立的关系模式

- 关系的属性：与该多值联系相连的各实体的码以及联系本身的属性
- 关系的码：各实体码的组合



转换成如下关系模式：

供应商(供应商代码, 供应商名称, 供应商地址, 供应商所在城市)

项目(项目代码, 项目名, 项目所在地)

零件(零件代码, 零件名称, 颜色, 重量)

供应(供应商代码, 项目代码, 零件代码)

(项目, 零件)

E-R图集成 (视图集成)

● 合并后的E-R图可能存在冗余：



由于Q1可以由Q2和Q3决定，故Q1多余，同时“使用”属性也可以看作“购买”和“消耗”传递表达“使用”关系，可以去掉依赖关系属性。另外，前两个的属性本身也是完全重叠的，应去重。

② 1:n 联系的转换

①



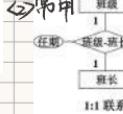
转换成如下关系模式：

班级(班号, 班级, 班主任, 入学时间)

班长(班号, 姓名, 性别, 年龄)

注：严格地说，如果一个班级在不同时间有不不同的班长，应是tn联系。龙虎只写为1:n。

②



转换成如下关系模式：

班级(班号, 班级, 班主任, 入学时间)

常用(班号, 书名, 借阅时间)

注：严格地说，如果一个班级在不同时间借阅不同的书籍，应是tn联系。龙虎只写为1:n。

④ n:m 联系的转换

① m:n 联系转换为一个关系模式

- 关系的属性：与该联系相关的各实体的码以及联系本身的属性
- 关系的码：各实体码的组合



转换成如下关系模式：

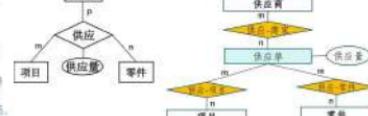
课程(课程号, 课程名, 学分, 先修课号)

学生(学号, 姓名, 性别, 年龄)

成绩(学生, 课程, 成绩)

② 分解为多个二元关系模式

● 联系供应商(项目, 供应商)是供应商(项目)、零件(项目)三个实体之间的联系。



代数优化算法

优化算法

输入：关系代数表达式的语法规则。

输出：计算表达式的优化程序。

方法步骤：

1. $\sigma_{F_1} \times F_2 \wedge \dots \wedge F_n (E)$ 转换为 $\sigma_{F_1} (\sigma_{F_2} (\dots (\sigma_{F_n} (E))))$;

2. 每个投影操作尽可能移向叶端；

3. 每个投影操作尽可能靠近叶端；

4. 将投影选择合并成一个选择后跟一个投影；

5. 语法规则节点分组；

6. 生成程序，每组节点为程序中的一步。

例：已知关系模式如下：

$S(Sno, Name, Sage, Ssex)$

$C(Cno, Cname, Teacher)$

$T(Tno, Tname, Tage, Tsex, Tgrade)$

查询至少有两个以上年龄的男教师的姓名和学号。

1. 写出关系表达式：

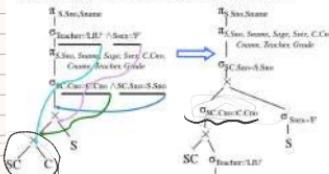
$\sigma_{T.Tsex = 'M'} \times \sigma_{T.Tage >= 20} \times \sigma_{T.Tage <= 30} (T)$

2. 用 $\sigma_{A \times B}$ 替换嵌套选择。得到如下表达式：

$\sigma_{S.Sno, S.Name} (\sigma_{Teacher -> T.Tno \times T.Tage >= 20 \times T.Tage <= 30} (T))$

其中 $L = S.Sno, S.Name, Sage, Ssex, C.Cno, Cname, Teacher, Grade$

4. 将投影项拆分，并利用等价规则将其尽可能移向叶端：



■ 事务是数据库恢复和并发控制的基本单位，是一个不可分割的工作单位

↓
ACID特性：原子性、一致性、隔离性、持久性

{
 事务故障
 系统故障
 行政故障
 其他故障
 }

二、数据恢复的方法

1. 静态转储（“冷备”）

在系统中有运行事务时进行，开始时数据处于一致性状态，期间不允许对运行事务的任何存取、修改。

优点：实现简单。

缺点：降低了数据的可用性。

2. 动态转储（“热备”）

能直接与用户事务并发进行。

转储期间允许对数据仓库进行存取、修改。

利用动态转储得到的后备副本进行故障恢复。

→ 建立日志文件，记录动态转储期间各事务对数据库的修改活动。

→ 利用后备副本+日志文件，进行数据库恢复。

优点：

不用等待正在运行的事务结束，且不影响新事务的运行，提高了数据库的可用性。

缺点：

单机依赖后备副本，无法保证副本中的数据正确有效。

日志文件 —— 以记录为单位

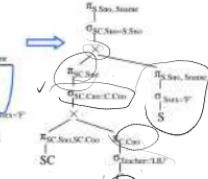
—— 以数据块为单位

各事务的开始标记 (begin transaction)

各事务的结束标记 (commit or rollback)

各事务的所有更新操作

7. 将新增投影在等价情况下尽可能移向叶端，结合随后的选择运算增加新投影；

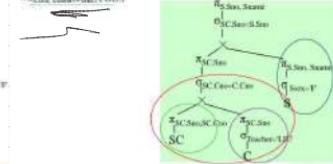


8. 对内结点分组原则：

- 每一贝祖运算 $\times, \bowtie, \cup, \cap$ 和它所有的直接祖先为一组(这些直接祖先为 σ 的逆运算)。
- 如果其后的连接到叶子全是由目运算，则也将它们并入该组，但当双目运算需要卡积积(*)时，而且其后的连接不能与它结合为等值连接时除外，把这些单目运算单独分组为一组。

9. 优化后的关系代数表达式：

$\pi_{S.Sno, S.name} (\sigma_{C.Cno = 'C1'} \times \sigma_{C.Cno = 'C2'} (\pi_{C.Cno, C.Cname} (SC \times C))) \times \sigma_{T.Tno = 'T1'} (\sigma_{T.Tage >= 20 \times T.Tage <= 30} (T))$



• 排它锁 (Exclusive lock, 简记为X锁, 又称为写锁)

若事务T对数据对象A加上X锁, 则只允许T读取和修改A, 其它任何事务都不能再对A加任何类型的锁, 直到T释放A上的锁。

• 共享锁 (Share lock, 简记为S锁, 又称为读锁)

若事务T对数据对象A加上S锁, 则其它事务只能再对A加S锁, 而不能加X锁。直到T释放A上的锁。

■ 锁的相容矩阵

T _i \ T _j	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

Y=Yes, 相容的请求
N=No, 不相容的请求

解决死锁:

- ① 预防死锁
② 检测与解除死锁

■ 两阶段锁 (Two-Phase Locking, 简称2PL) 协议

- 事务分为两个阶段

第一阶段是获得封锁, 也称为扩展阶段。
第二阶段是释放封锁, 也称为收缩阶段。

■ 什么是可串行化调度?

多个事务的并发执行是正确的, 当且仅当其结果与按某次序串行地执行这些事务时的结果相同, 称这种调度策略为可串行化调度。

并发执行是正确的 \Leftrightarrow 该调度可串行化调度。

→ 不可串操作进行交换顺序

冲突操作: 是指不同事务对同一个数据的读写操作和写写操作,

R(x)与W(y) // 事务T_i读x, 事务T_j写y

W(x)与W(y) // 事务T_i写x, 事务T_j写y

其他操作是不冲突操作。

■ 串行化调度

- 串行, 并发低, 时延高

■ 冲突可串行化调度

- 并行, 并发高, 时延低

- 事后检测

- 事前可保证 \rightarrow 两阶段锁 (2PL)

冲突可串行化调度的验证复杂度较低, 而且可以通过阶段锁封锁2PL实现。因此是数据库中通常采用的一种方法。

冲突可串行化调度一定是可串行化调度。
(其结果一定是正确的)。反之, 则不然。



① SQL 语句

尚在等候 SMALLINT CHECK (尚在等候 BETWEEN 1 AND 10).

DATE 类型

-- 查询大于指定日期的数据
SELECT * FROM orders
WHERE order_date > '2022-01-01';

-- 查询在指定日期范围内的数据
SELECT * FROM orders
WHERE order_date BETWEEN '2022-01-01' AND '2022-12-31';

-- 查询比指定日期小并且不等于指定日期的数据
SELECT * FROM orders
WHERE order_date < '2022-01-01' AND order_date != '2021-12-31';

BETWEEN .. AND ..

CHAR(n)

VARCHAR(255)

INT

SMALLINT

BIGINT

FLOAT(n)

DATE

YYYY-MM-DD

TIME

HH:MM:SS

② 并发控制

Xlock(A)

:

Xlock(A)

等待

等待

Unlock(A)

Xlock(A);

重新申请

X=Read(A)

1) C → AD

2) C → A, C → D 分解规则

3) 由 C ⊑ AC 放

AC → C 自反律

4) AC → D

③ AC → B 被 F 逻辑蕴含判定

解：求 $(AC)^+ = ACDE$

则由 B 不是 $(AC)^+$ ，故不被 F 逻辑蕴含

2. 用关系代数表达式，查询读者“张三”借阅“数据库系统”书籍的日期与天数。

解： $\pi_{Date, Interval}(BR \bowtie_{B.Rno=Reader.Rno} \{Reader\name = '张三'\} \bowtie_{Book.Bno=BR.Bno} \{Book\name = '数据库系统'\})$

6. 用 SQL 语句，查询书籍编号 B0010 的读者借阅情况，输出每位读者编号、姓名、总的借阅天数（一位读者可以多次借阅该书），按所总的借阅天数降序排列。

解：SELECT BR.Rno, MIN(Rname), SUM(Interval)
FROM Reader, BR
WHERE Reader.Rno=BR.Rno AND BR.Bno= 'B0010'
GROUP BY BR.Rno
ORDER BY SUM(Interval) DESC;

解：CREATE VIEW IS_BR(Rno,Bname,avginterval)

AS
SELECT BR.Rno,MIN(Bname),AVG(Interval) avginterval
FROM BR,Book

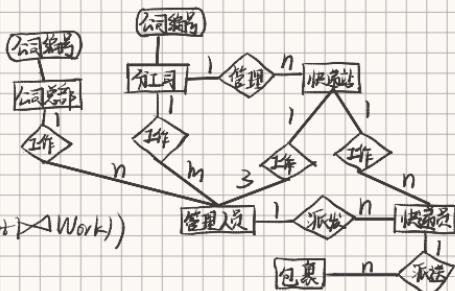
WHERE BR.Bno=Book.Bno
GROUP BY BR.Bno;

5. 用 SQL 语句查询没有借阅任何书籍的读者信息：

解法一：SELECT *
FROM Reader
WHERE NOT EXISTS
(SELECT *
FROM BR
WHERE BR.Rno=Reader.Rno);

解法二：SELECT *

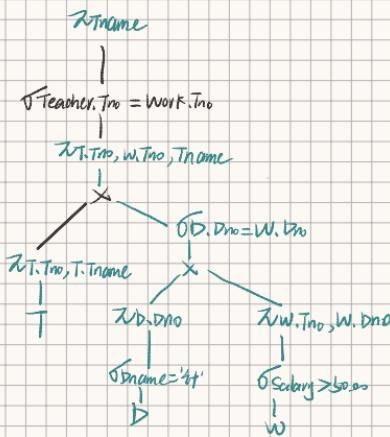
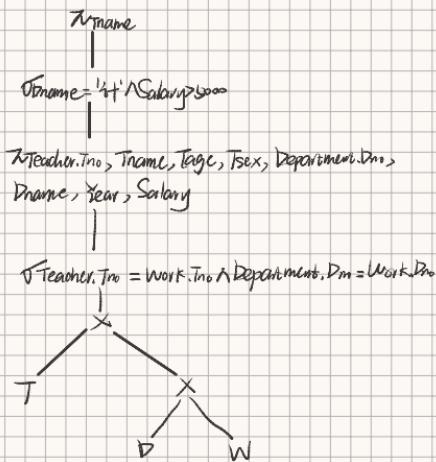
FROM Reader
WHERE Rno IN
(SELECT Rno
FROM Reader
EXCEPT
SELECT Rno
FROM BR);



$\pi_{Tname}(\sigma_{Dname='it'} \wedge \text{Salary} > 5000 (\text{Teacher} \bowtie \text{Department} \bowtie \text{Work}))$

替换 $\pi_{Tname}(\sigma_{Dname='it'} \wedge \text{Salary} > 5000 (\forall_L (\exists_{Teacher.Tno = Work.Tno} \wedge \text{Department.Dno} = \text{Work.Dno} (\text{Teacher} \times \text{Department} \times \text{Work}))))$

查询表达式:



- 数据库 (DataBase, 简称 DB) : 长期存储在计算机内、有组织、可共享的数据集合。



数据库管理系统 (DataBase Management system, 简称 DBMS) : 专门用于管理数据库的软件。

包括: 相互关联的数据集合、用以访问这些数据的程序组成。



● 数据库管理系统软件的出现使得数据存储、数据管理与数据应用分离。

● 数据库管理系统采用外模式 - 模式 - 内模式 **三级模式**, 外模式 / 模式和模式 / 内模式 **两级映像结构**来实现。

数据模型

● 是数据及其联系在计算机中的表示和组织形式的描述。

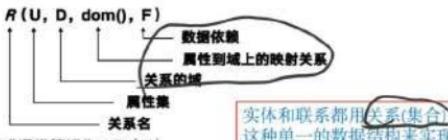
● 组成三要素:

- { ① 数据结构
- ② 数据操纵
- ③ 数据完整性约束



- 建立在严格的数学基础
- 关系是规范化的
- 关系的分量是不可分割的数据项

● 关系模式(Relation Schema), 即关系的型的定义



● 关系模式通常简记为: R (U)

关系语言特点:

关系语言是一种高度非过程化的语言(?)

✓ 关系语言是完备的

什么是关系代数?

关系代数(Relational Algebra)以集合为基础, 定义一组运算, 由已知关系经过一列运算, 得到需要的查询结果(新的关系)的表达方式。

关系代数是过程化的查询语言。

关系的完整性分为:

- 1) 实体完整性
- 2) 参照完整性
- 3) 用户定义完整性

SQL的特点:

- (1) 综合统一
集DDL、DML、DCL于一体。
- (2) 高度非过程化
不用考虑如何实现, 只需提出“做什么”, 不关心“怎么做”。
- (3) 面向集合的操作方式
查询、插入、删除、修改的操作对象及结果都是集合。

数据定义语言DDL 用于定义和修改数据库结构, 提供创建、修改、删除关系模式、索引、视图的命令

数据操纵语言DML 用于查询和更新数据, 提供从数据库中查询数据, 以及插入数据、修改数据、删除数据的命令

数据控制语言DCL 用于创建用户角色和权限, 以及控制数据库访问

- Student关系模式不是一个“好”的模式。
- 所谓“好”的模式: 不会发生插入异常、删除异常、更新异常, 数据冗余应尽可能少。
- 原因: 是模式中属性间存在某些依赖关系引起的。
 - 如: Mname的取值依赖于Sdept的值。
- 解决方法: 通过**分解关系模式**来消除其中不合适的依赖。

规范化(Normalization)是指定义一组关系模式应该符合的条件(范式), 而符合这些条件的关系模式就不存在某些操作异常, 允许也会减小。

● 数据库规范化设计具有以下几个优点:

- (1) **降低数据存储和维护数据一致性的成本**, 即减少冗余数据, 减少存储空间的开销; 当数据发生更新(以及插入和删除)的时候, 不必同时更新那些冗余数据, 降低了维护数据一致性的成本。
- (2) **便于设计合理的关系表间的依赖和约束关系**, 便于实现数据完整性和一致性, 避免插入异常、删除异常、更新异常。
- (3) **便于设计合理的数据库结构**, 以提高数据库系统的整体性能。

● **数据库是组织、存储和管理数据的集合**

```

CREATE TABLE SC
  (Sno CHAR(9) NOT NULL, /*NOT NULL可以省略*/
   Cno CHAR(4) NOT NULL, /*NOT NULL可以省略*/
   Grade SMALLINT,
   PRIMARY KEY (Sno, Cno),
   FOREIGN KEY (Sno) REFERENCES Student(Sno)
     ON DELETE CASCADE /*级联删除SC表中相应的元组*/
     ON UPDATE CASCADE /*级联更新SC表中相应的元组*/
   FOREIGN KEY (Cno) REFERENCES Course(Cno)
   ON DELETE NO ACTION
   /*删除course表中的元组造成与SC表不一致时拒绝删除*/
   ON UPDATE CASCADE
   /*更新course表中的cno时, 该联更新SC表中相应的元组*/
  
```

事务是一系列的数据库操作，是数据库应用程序的基本逻辑单元。

事务(Transaction)是用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位。

事务是数据库恢复和并发控制的基本单位

(1) COMMIT语句表示：事务正常结束，提交事务的所有操作
(读+更新) 事务中所有对数据库的更新写回磁盘上的物理数据
库称，永久生效

(2) ROLLBACK语句表示：事务异常终止，事务运行的过程中发生了故障，不能继续执行、撤销(回滚)事务的所有更新操作，使事务滚回到开始时的状态

■原子性 (Atomicity) ■一致性 (Consistency)

事务中包括的诸操作要么都做，要么都不做。
事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。

■隔离性 (Isolation)

一个事务的执行不能被其他事务干扰。

即，每个事务的执行效果与系统中只有该事务的执行效果一样。

事务的隔离性与事务并发执行紧密相关。

■持久性 (Durability)

一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其执行结果有任何影响。

故障恢复保证事务的原子性(A)和持久性(D)。

并发控制保证事务的一致性(C)和隔离性(I)。

故障的种类主要有

1. 事务故障 ★

2. 系统故障 ★

3. 介质故障

4. 其他故障

什么是事务故障

事务在运行过程中未运行至正常终止点而夭折了。

常见原因

- 输入数据有误
- 运算溢出
- 违反了某些完整性限制
- 某些应用程序出错
- 并行事务发生死锁
- 事务故障恢复

(1) 可见原因：由事务程序本身产生

(2) 不可见原因：由DBMS强行回滚该事务[MIT]

例：银行转账事务的代码

```
BEGIN TRANSACTION  
读取账户甲的余额BALANCE;  
BALANCE = BALANCE - AMOUNT;  
IF (BALANCE < 0) THEN  
    [打印‘余额不足，不能转账’];  
    ROLLBACK;  
ELSE  
    {读账户乙的余额BALANCE};  
    BALANCE += BALANCE1 *  
    AMOUNT;  
    [写回BALANCE];
```

恢复步骤：

- (1) 反向扫描文件日志(从后向前)，查找该事务的更新操作。
- (2) 对该事务的更新操作执行逆操作(将“更新前的值”写入数据库。对插入执行删除，对删除执行插入，对修改执行修改)。
- (3) 继续反向扫描日志文件，查找该事务的其他更新操作，执行2。
- (4) 扫描到该事务的开始标记，事务故障恢复结束。

什么是介质故障

存储数据库的设备发生故障，使存储在其上的数据部分或全部受到破坏丢失，又称磁盘故障。

常见原因

- 磁盘损坏
- 磁头碰撞
- 操作系统的潜在错误
- 瞬时强磁场干扰
- 介质故障恢复

(1) DBA做好数据库备份计划。

(2) 装入发生介质故障前某个时刻的数据库副本。

(3) 重做该时刻前的所有成功事务，提交结果记入数据库。

什么是系统故障

整个系统的正常运行突然被破坏

所有正在运行事务全部非正常终止

内存中缓冲区信息全部丢失，外存中数据未受影响

常见原因

- 操作系统、DBMS代码错误
- 硬件错误，如CPU故障
- 突然断电

系统故障恢复

(1) 回滚非正常终止的事务

(2) 重做缓冲区中已提交的事务

恢复操作的基本原理是冗余，即利用存储在系统其它地方的冗余数据来重建数据库中已被破坏或不正确的那部分数据。

恢复机制涉及的关键问题

(1) 建立冗余数据

- 数据转储(dump)
- 日志文件(log)：记录数据库系统变更的历史信息。

一、什么是数据转储

DBA定期将整个数据库复制到磁盘或其他存储介质上保存的过程。

这些备用数据称为后备副本(Backup)或后援副本。

二、数据转储的方法

1. 静态转储(“冷备”)

在系统中无运行事务时进行。开始时数据库处于一致性状态，期间不允许对数据库的任何存取、修改。

优点：实现简单。

缺点：降低了数据库的可用性。

2. 动态转储（“热备”）

- 转储操作与用户事务并发进行

转储期间允许对数据库进行存取、修改。

利用动态转储得到的后备副本进行故障恢复。

✓ 建立日志文件，记录动态转储期间各事务对数据库的修改活动。

✓ 利用后备副本+日志文件，进行数据库恢复。

■ 优点

不用等待正在运行的事务结束，且不影响新事务的运行，提高了数据库的可用性。

■ 缺点

单纯依靠后备副本，无法保证副本中的数据正确有效。

五、登记日志文件原则

为保证数据库是可恢复的，必须遵循两条原则：

(1) 登记次序严格按并行事务执行的时间次序；

(2) **先写日志文件，后写数据库：**

写日志文件：把修改的日志记录写到日志文件

写数据库操作：把对数据的修改写到数据库中

3. 全量转储和增量转储

- 全量转储：每次转储全部数据库

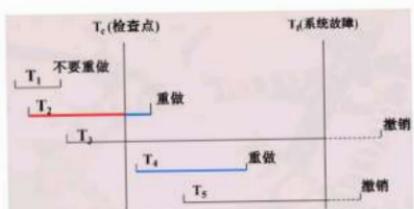
- 增量转储：只备份上次备份后更新过的数据。

- 全量备份与增量备份比较

· 从恢复角度看，使用全量备份得到的后备副本进行恢复往往更方便。

· 如果数据库很大，事务处理又十分频繁，则增量备份方式更实用、更有效。

基于检查点的系统故障恢复



系统出现故障时，恢复子系统将根据事务的不同状态采取不同的恢复策略。

3) 介质故障的恢复

恢复步骤：

(1) 装入最新的后备数据库副本；

· 装入静态转储的数据库副本。数据库即处于一致性状态

· 对于装入动态转储的数据库副本，还须同时装入转储时刻的日志文件副本。利用恢复系统故障的方法，将数据库恢复到一致性状态。

(2) 装入日志文件副本，重做已完成的事务。

■ 并发操作带来的数据不一致性

(1) 丢失修改 (lost update)

(2) 不可重复读 (non-repeatable read)

(3) 读“脏”数据 (dirty read)

冲突操作：是指不同事务对同一个数据的读写操作和写写操作。

R(x)与W(x) /* 事务T1读x，事务T2写x */

W(x)与W(x) /* 事务T1写x，事务T2写x */