# COMP9311 考点复习

UNSW COMP9311

---

**Outline** egP583/P304 指原书583页/中文书304页；如果只有 /P304 指中文书304页。

https://courses.cs.washington.edu/courses/cse414/14wi/exams/
如果下文有什么错误请联系159595223@qq.com 修改，万分感谢

祝大家考出好成绩

- 6月13 新增
  Q1 判断
- **Course_name** ：COMP9311 Database；
- **Lecturer** ：XUEMIN LIN
- **Author** ： ZESHI WU

---


COMP9311 考点复习
Q1 判断
Q2 DB design
2.1 ER图
2.2 ER映射表格 ER to Relational Data Model Mapping
Q3 Relational Algebra & FD范式
概念性的东西
3.1 Relational Algebra(PPT2.2)
3.2 Normal Forms 知识点
3.3 FD function dependency知识点
3.3.1 侯选建Candidate Key
3.4 无损连接分解 lossless-joint
3.4.1 check whether lossless-joint
3.4.2 lossless decomposition into 3NF

3.4.3 lossless decomposition into BCNF（答案不唯一）
3.5 保持函数依赖分解 dependency-preserving
3.5.1 check whether dependency—preserving
3.5.2 dependency—preserving decomposition into 3NF
3.5.3 dependency—preserving decomposition into BCNF
3.6 dependency—preserving & lossless decomposition 其实3.4/3.5 都可以用以下解法
Fm
dependency—preserving & lossless decomposition
Q4 Transections & Indexes
4.1 概念*
Q4 Transections & Indexes
4.1 概念（没时间可以不看）
4.1.1 Transaction（2-5）, Concurrency（6-7）, Recovery（8—）
4.1.2 transection 造成的影响
4.1.3 schedules
4.1.4 串行化 & 可串行性
4.1.5 ACID
4.1.6 基于锁的并发控制
4.1.6.1 2PL
4.1.6.2 时间戳timestamp order
4.1.7数据库的隔离级别*
4.1.8 事务恢复 / p41
4.1.9 基于延时更新的恢复（不想考虑这么多 可以照着ppt 的思路做）
4.1.10 基于及时更新的恢复
4.1.n 概念考题 判断，简答6]
4.1.8 概念考题
4.2 冲突调度-redo，undo
4.3 判断死锁
4.4 index
4.4.1 primary index
4.4.2 secondary index
4.4.3 cluster/uncluster index

考题
Q5 空间数据库

**非考点**

- PLpgSQL, SQL but we may ask yes or no questions in the exam
- Multi-versioning and Optimistic Concurrency Control
- Graph DB (week 10 and 11)

**考点**

- Q1: Yes or No (20 marks); Correct: 2 marks, wrong: -1 mark.
- Q2: DB design – ER and Relational DB (20 marks)
- Q3: Relational Algebra & FD & NFs (30 marks)
- Q4: Transactions & Indexes (20 marks)
- Q5: Spatial Data: use relational algebra to answer a Q (10 marks)

# Q1 判断

看所有的ppt， 几题sql的结果判断题，基本都是考基础概念题

- A materialized view may contain data that is not up to date ( T )
- A query that uses a virtual view always runs much faster than the same query using a materialized view. ( F )
- An index is special case of a virtual view. ( F )
- An index is a special case of a materialized view. ( T )
- In a static database, strict two phase locking guarantees that the schedule is serializable while two phase locking does not. ( F )
- Strict two phase locking guarantees that the schedule is recoverable, while two phase locking does not. ( T )
- In a dynamic database, strict two phase locking can prevent phantoms, while two phase locking cannot. ( F: needs table lock )
- Strict two phase locking holds all the locks until the end of a transaction, while two

phase locking may release the locks earlier.（T）

# Q2 DB design

## 2.1 ER图

- 画图要点/p54

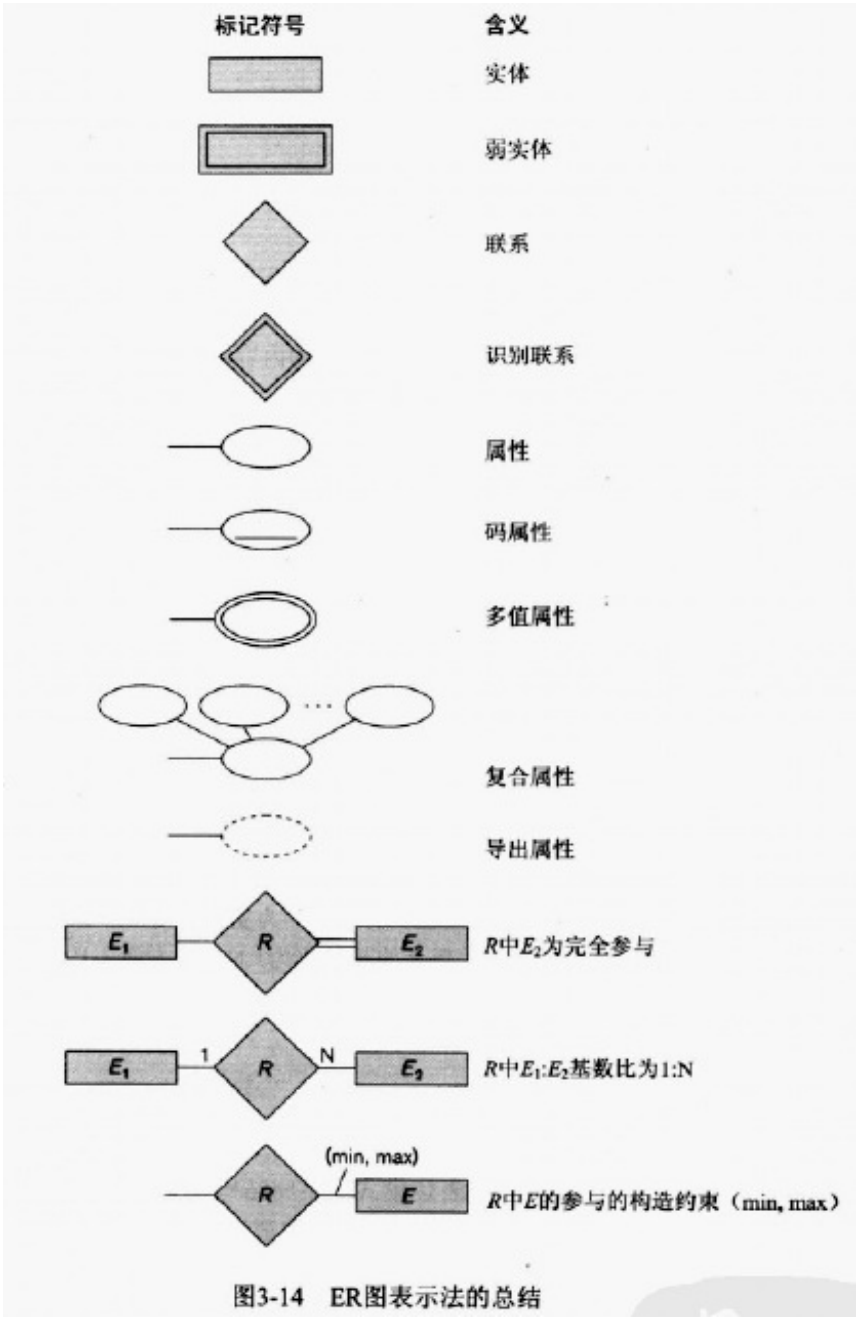


图3-14 ER图表示法的总结

- 总结要点

ER model is popular for doing conceptual design

- high-level, models relatively easy to understand
- good expressive power, can capture many details

Basic constructs: entities, relationships, attributes

Relationship constraints: total / partial, n:m / 1:n / 1:1

Other constructs: ineritance hierarchies, weak entities

Many notational variants of ER exist
(especially in the expression of constraints on relationships)

## 2.2 ER映射表格 ER to Relational Data Model Mapping

- PPT 2.1
- 一共7步：
  - 1.实体 E
    - Attributes: 所以单一属性，除了多值属性
    - Key：选一个当主键
  - 2.弱实体 W
    - 所有单一属性
    - key: Foreign Key（W依赖的实体E的主键，**箭号指向E的主键**） + partial key（W的主键） **两个都要有下划线**
  - 3.找1:1关系（实体S、T）
    - 关系带有的属性append 在实体T（如果T完全参与）后面，
    - 在T的关系后写一个S的主键作为外键（**箭号指向S的主键**）
  - 4.找1：N关系
    - 如果T（完全参与），append在T的关系后，并将S的主键作为外键写在T后面（**箭号指向S的主键**）
  - 5.找M:N关系
    - 新建一个表

        - 把S、T的主键都作为新表的外键（**两个箭号指向S、T的主键**）
        - **两个都要有下划线**
    - 6.多值属性
        - 新建一个表
        - 将该属性隶属于的实体E的主键作为外键（**箭号指向E的主键**）
        - **所有的属性都要下划线**
    - 7.第七步 如果又多个关系，一个菱形联系 连接多个实体
        - 参照5 是不是不需要了 忘了

# Q3 Relational Algebra & FD范式

## 概念性的东西

- 超键(super key):在关系中能唯一标识元组的属性集称为关系模式的超键
- 候选键(candidate key):不含有多余属性的超键称为候选键
- 主键(primary key):用户选作元组标识的一个候选键程序主键
- 例子：身份证、姓名、性别、年龄：
    - superkey：
        - 身份证 唯一 所以是一个超键
        - 姓名 唯一 所以是一个超键
        - （姓名，性别） 唯一 所以是一个超键
        - （姓名，性别，年龄） 唯一 所以是一个超键
        - 这里可以看出，超键的组合是唯一的，但可能不是最小唯一的
    - candidatekey：
        - 身份证 唯一而且没有多余属性 所以是一个候选键
        - 姓名 唯一而且没有多余属性 所以是一个候选键
        - 这里可以看出，候选键是没有多余属性的超键
    - primary key
        - 考虑输入查询方便性，选择身份证为主键
        - 也可以考虑习惯，选择姓名为主键
        - 主键是选中的一个候选键
- 一题搞懂什么是候选键：

- 在SQL Server数据库中，有一个学生信息表如下所示，在该表中不能作为候选键的属性集合为（ ）（选择一项）
  学号 姓名 性别 年龄 系别 专业
  20020612 李辉 男 20 计算机 软件开发
  20060613 张明 男 18 计算机 软件开发
  20060614 王小玉 女 19 物理 力学
  20060615 李淑华 女 17 生物 动物学
  20060616 赵静 男 21 化学 食品化学
  20060617 赵静 女 20 生物 植物学
  a ) {学号}
  b ) {学号、姓名}
  c ) {年龄、系别}
  d ) {姓名、性别}
  e ) {姓名、专业}
    - 透过概念，我们可以了解到，超键包含着候选键，候选键中包含着主键。主键一定是惟一的。为什么呢？因为他的爷爷超键就是惟一的。
    - abcde5个答案都可以作为超键，他们组合在一起的集合可以用来惟一的标识一个实体。
    - 候选键要求是不能包含多余属性的超键，我们看一下答案b。在答案b中，如果我们不使用姓名也可以惟一的标识一条数据实体，可以说姓名字段在这里是多余的。那么很明显，b选项包含了多余字段属性。那么这题答案应该选择b。

## 3.1 Relational Algebra(PPT2.2)

- **Select**： $\sigma_{(name=1)AND\ NOT(...)}$（$ERNOLMENT$）
  - using AND, OR and NOT.
- **Project:** $\pi_{(Department)}(\sigma_{(Department)}$（$ERNOLMENT$）$)$
- **Union：** R1 ∪ R2
  - $\pi(R1 \cup R2) = \pi(R1) \cup \pi(R2)$
- **Intersection：** R1 ∩ R2
  - $\pi(R1 \cap R2) \neq \pi(R1) \cap \pi(R2)$
- **Difference：** R-S
- **Cartesian Product笛卡尔积** $R \times S$

- 又称Cross Product叉积, 最后结果有$n_R * n_S$元组
- /p120

  $\text{FEMALE\_EMPS} \leftarrow \sigma_{\text{Sex='F'}}(\text{EMPLOYEE})$

  $\text{EMPNAMES} \leftarrow \pi_{\text{Fname, Lname, Ssn}}(\text{FEMALE\_EMPS})$
- $\text{EMP\_DEPENDENTS} \leftarrow \text{EMPNAMES} \times \text{DEPENDENT}$

  $\text{ACTUAL\_DEPENDENTS} \leftarrow \sigma_{\text{Ssn=Essn}}(\text{EMP\_DEPENDENTS})$

  $\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Dependent\_name}}(\text{ACTUAL\_DEPENDENTS})$

**FEMALE_EMPS**

| Fname | Minit | Lname | Ssn | Bdate | Address | Sex | Salary | Super_ssn | Dno |
|---|---|---|---|---|---|---|---|---|---|
| Alicia | J | Zelaya | 999887777 | 1968-07-19 | 3321Castle, Spring, TX | F | 25000 | 987654321 | 4 |
| Jennifer | S | Wallace | 987654321 | 1941-06-20 | 291Berry, Bellaire, TX | F | 43000 | 888665555 | 4 |
| Joyce | A | English | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX | F | 25000 | 333445555 | 5 |

**EMPNAMES**

| Fname | Lname | Ssn |
|---|---|---|
| Alicia | Zelaya | 999887777 |
| Jennifer | Wallace | 987654321 |
| Joyce | English | 453453453 |

**EMP_DEPENDENTS**

| Fname | Lname | Ssn | Essn | Dependent_name | Sex | Bdate | ... |
|---|---|---|---|---|---|---|---|
| Alicia | Zelaya | 999887777 | 333445555 | Alice | F | 1986-04-05 | ... |
| Alicia | Zelaya | 999887777 | 333445555 | Theodore | M | 1983-10-25 | ... |
| Alicia | Zelaya | 999887777 | 333445555 | Joy | F | 1958-05-03 | ... |
| Alicia | Zelaya | 999887777 | 987654321 | Abner | M | 1942-02-28 | ... |
| Alicia | Zelaya | 999887777 | 123456789 | Michael | M | 1988-01-04 | ... |
| Alicia | Zelaya | 999887777 | 123456789 | Alice | F | 1988-12-30 | ... |
| Alicia | Zelaya | 999887777 | 123456789 | Elizabeth | F | 1967-05-05 | ... |
| Jennifer | Wallace | 987654321 | 333445555 | Alice | F | 1986-04-05 | ... |
| Jennifer | Wallace | 987654321 | 333445555 | Theodore | M | 1983-10-25 | ... |
| Jennifer | Wallace | 987654321 | 333445555 | Joy | F | 1958-05-03 | ... |
| Jennifer | Wallace | 987654321 | 987654321 | Abner | M | 1942-02-28 | ... |
| Jennifer | Wallace | 987654321 | 123456789 | Michael | M | 1988-01-04 | ... |
| Jennifer | Wallace | 987654321 | 123456789 | Alice | F | 1988-12-30 | ... |
| Jennifer | Wallace | 987654321 | 123456789 | Elizabeth | F | 1967-05-05 | ... |
| Joyce | English | 453453453 | 333445555 | Alice | F | 1986-04-05 | ... |
| Joyce | English | 453453453 | 333445555 | Theodore | M | 1983-10-25 | ... |
| Joyce | English | 453453453 | 333445555 | Joy | F | 1958-05-03 | ... |
| Joyce | English | 453453453 | 987654321 | Abner | M | 1942-02-28 | ... |
| Joyce | English | 453453453 | 123456789 | Michael | M | 1988-01-04 | ... |
| Joyce | English | 453453453 | 123456789 | Alice | F | 1988-12-30 | ... |
| Joyce | English | 453453453 | 123456789 | Elizabeth | F | 1967-05-05 | ... |

**ACTUAL_DEPENDENTS**

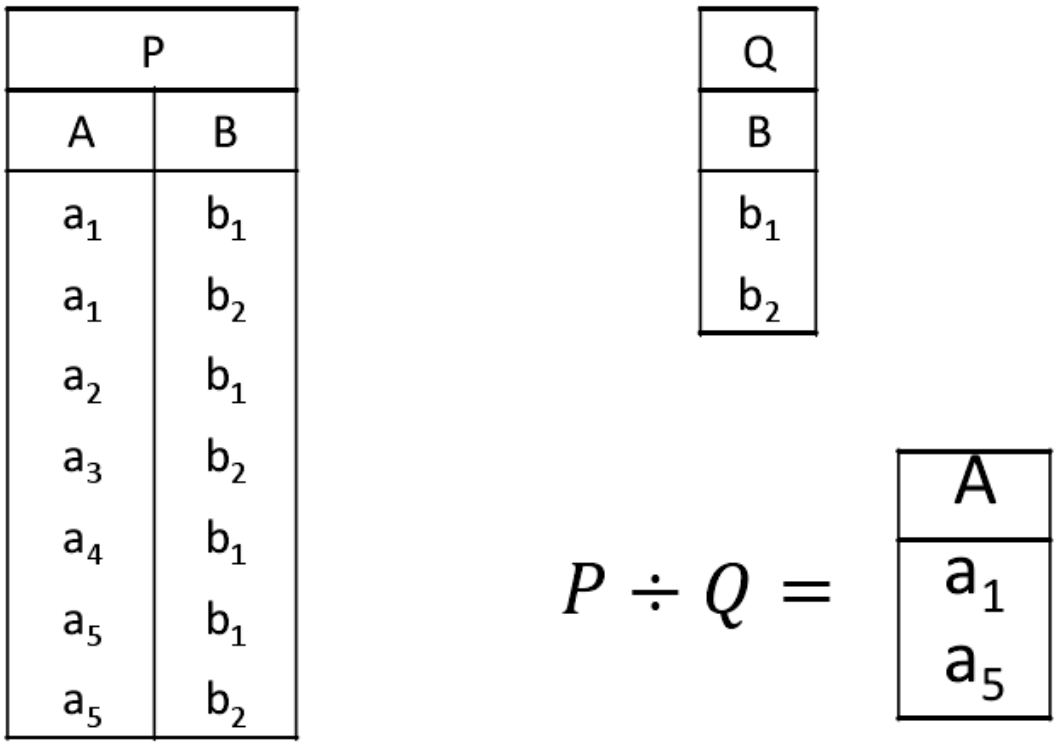
| Fname | Lname | Ssn | Essn | Dependent_name | Sex | Bdate | ... |
|---|---|---|---|---|---|---|---|
| Jennifer | Wallace | 987654321 | 987654321 | Abner | M | 1942-02-28 | ... |

**RESULT**

| Fname | Lname | Dependent_name |
|---|---|---|
| Jennifer | Wallace | Abner |

图6-5 笛卡儿积（叉积）操作

- **Join：** $R \bowtie_B S$
  - 等价于上面的图片 $EMPNAMES \bowtie_{(Ssn=Essn)} DEPENDENT$
  - EQUI JOIN 等值连接，有=
  - NATUAL JOIN 因为Ssn=Essn 中有一个值是多余的，所以引入自然连接

    - $R \bowtie_{(Supervisor),(Person)} S$
    - 书上是用*，以课件为主
- **Divide:** $R \div S$
    - 

| P | |
|---|---|
| A | B |
| $a_1$ | $b_1$ |
| $a_1$ | $b_2$ |
| $a_2$ | $b_1$ |
| $a_3$ | $b_2$ |
| $a_4$ | $b_1$ |
| $a_5$ | $b_1$ |
| $a_5$ | $b_2$ |

| Q |
|---|
| B |
| $b_1$ |
| $b_2$ |

$$P \div Q =$$

| A |
|---|
| $a_1$ |
| $a_5$ |

    - 检索 小明参与所有工作项目中所有雇员的名字
    - $R \div S$ 等价于 $$T1 \leftarrow \pi_Y(R)$$ $$T2 \leftarrow \pi_Y((S \times T1) - R)$$ $$T \leftarrow T1 - T2$$
- 考题：
    - https://wenku.baidu.com/view/51a199f8cc22bcd126ff0cc7.html
    - https://wenku.baidu.com/view/2206a14fa8114431b90dd85a.html
    - https://wenku.baidu.com/view/31154cc09ec3d5bbfd0a7412.html?from=search

## 3.2 Normal Forms 知识点

- 函数依赖
    - 对于R上的任意两个关系r1,r2, 若，则称X决定Y或者Y依赖于X,表示为$X \rightarrow Y$

- *其实就是若X为定义域,能否得到唯一的Y*

- 闭包
  - 不会解释。。直接看例子
  - $R = \{a \to b, a \to c, a \to d, d \to e\}$
  - a的闭包$a^+ = \{a, b, c, d, e\}$
  - 考题: 求以下关系的主键(可以推导出属性集的所有属性)
    - 1. $FD = \{A \to B, C \to D, E \to FG\}$
      2. $FD = \{A \to B, B \to C, C \to D\}$
      3. $FD = \{A \to B, B \to C, C \to A\}$
      4. $FD = \{ABH \to C, A \to D, C \to E\ F \to A, E \to F, BGH \to E\}$
    - ACE
    - A
    - A or B or C
    - BGH
- 第几范式NF (COPY FROM YUNQIU XU)
  - 1NF
    - 每个属性都不可再分, 即不允许嵌套表
    - 不符合第一范式的栗子
    - **indivisible value**
    - | 编号 | 品名 | 进货 | | 销售 | | 备注 |
      |---|---|---|---|---|---|---|
      | | | 数量 | 单价 | 数量 | 单价 | |
      | | | | | | | |
  - 2NF
    - 在符合第一范式的基础上,非主属性完全函数依赖于主属性, 即不允许partial dependencies存在

不符合第二范式的栗子

| 学生 | 课程 | 老师 | 老师职称 | 教材 | 教室 | 上课时间 |
|---|---|---|---|---|---|---|
| 小明 | 一年级语文（上） | 大宝 | 副教授 | 《小学语文1》 | 101 | 14：30 |

一个学生上一门课，一定在特定某个教室。所以有（学生，课程）->教室

一个学生上一门课，一定是特定某个老师教。所以有（学生，课程）->老师

一个学生上一门课，他老师的职称可以确定。所以有（学生，课程）->老师职称

- 一个学生上一门课，一定是特定某个教材。所以有（学生，课程）->教材

一个学生上一门课，一定在特定时间。所以有（学生，课程）->上课时间

因此（学生，课程）是一个码。

- 但是这里非主属性教材仅仅依赖于课程:对于同一门课, 不同学生需要的教材是一样的
- 导致的后果: 一旦需要修改教材,所有包含这门课程的数据(所有学生)都要进行修改
- 拆分为以下形式:

（学生，课程，老师，老师职称，教室，上课时间）和（课程，教材）

- 违反的时候回答：non—prime attribute X is funtionally determined by Y

- 3NF
  - 第二范式的基础上, 不允许非主属性通过传递依赖主属性
  - 还是上面的栗子: 老师职称依赖于老师, 老师依赖于主属性(学生,课程)
  - 拆分为以下形式: 老师变为主属性, 老师职称直接依赖于老师
  - （学生，课程，老师，教室，上课时间）和（课程，教材）和（老师，老师职称）
  - 违反的时候回答：non-prime attribute X of R is transitively dependent on the non-primary key.
- BCNF
  - 第三范式的基础上, 不允许主属性部分依赖或传递依赖于主属性
  - 违反的时候回答：there is a trivial functional dependency X → A holds in R, then A is a primary key of R & X is not superkey.
  - BCNF
  - 第三范式的基础上, 不允许主属性部分依赖或传递依赖于主属性
- 范式检查

- 


## 3.3 FD function dependency知识点

- 算闭包的时候要涉及到FD的知识，
- Armstrong’ s axioms

F1. Reflexivity e.g. $X \rightarrow X$

- a formal statement of *trivial dependencies*; useful for derivations

F2. Augmentation e.g. $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$

- if a dependency holds, then we can freely expand its left hand side

F3. Transitivity e.g. $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

- the "most powerful" inference rule; useful in multi-step derivations

F4. Additivity e.g. $X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$

- useful for constructing new right hand sides of *fd*s (also called union)

F5. Projectivity e.g. $X \rightarrow YZ \Rightarrow X \rightarrow Y, X \rightarrow Z$

- useful for reducing right hand sides of *fd*s (also called decomposition)

F6. Pseudotransitivity e.g. $X \rightarrow Y, YZ \rightarrow W \Rightarrow XZ \rightarrow W$

- shorthand for a common transitivity derivation

- 考题 参见 候选键， 应该不会考check whether FD，太简单

### 3.3.1 侯选建Candidate Key

- 候选键为可推导出所有依赖属性的主键集合, 但这个集合的真子集无法推导出全部依赖属性
- 解法：
  - R属性: 从未被依赖, 只出现在右边,肯定不是候选键一部分
  - L属性: 从未依赖于其他属性, 只出现在左边, 该属性肯定是某个候选键一部分(但不是所有)
  - N属性: 两边都没出现的属性, 一定会出现在所有候选键中
  - LR: 其他属性, 在两边都出现过, 逐个和以上一定会出现在候选键中的属性结合, 求闭包

- 第一步：先求出L、LR、N元素
- 第二步：判断N集合是否为空，如果为空，求出L的所有子集Sub，遍历Sub，求Key。若Key值为空，将L和LR集合做交运算，并将结果存在L中，再求出L的所有子集Sub，然后遍历Sub，求出Key。
- 第三步：当N集合不为空时，L和N集合做交运算，结果放在L中，求出L的所有子集Sub，再遍历Sub，求出Key，若Key值为空，L集合和LR集合做交运算，结果存放于L中，求出L的所有子集Sub，再遍历Sub，求出Key。

- 考题：

i. $C \rightarrow D,\ C \rightarrow A,\ B \rightarrow C$

[hide answer]

a. Candidate keys: *B*
b. Not BCNF ... e.g. in $C \rightarrow A$, *C* does not contain a key
c. Not 3NF ... e.g. in $C \rightarrow A$, *C* does not contain a key, *A* is not part of a key

ii. $B \rightarrow C,\ D \rightarrow A$

[hide answer]

a. Candidate keys: *BD*
b. Not 3NF ... neither right hand side is part of a key
c. Not BCNF ... neither left hand side contains a key

iii. $ABC \rightarrow D,\ D \rightarrow A$

[hide answer]

a. Candidate keys: *ABC BCD*
b. 3NF ... $ABC \rightarrow D$ is ok, and even $D \rightarrow A$ is ok, because *A* is a single attribute from the key
c. Not BCNF ... e.g. in $D \rightarrow A$, *D* does not contain a key

iv. $A \rightarrow B,\ BC \rightarrow D,\ A \rightarrow C$

[hide answer]

a. Candidate keys: *A*
b. Not 3NF ... e.g. in $A \rightarrow C$, *C* is not part of a key
c. Not BCNF ... e.g. in $BC \rightarrow D$, *BC* does not contain a key

v. $AB \rightarrow C,\ AB \rightarrow D,\ C \rightarrow A,\ D \rightarrow B$

[hide answer]

a. Candidate keys: *AB BC CD AD*
b. 3NF ... for *AB* case, first two fd's are ok, and the others are also ok because the RHS is a single attribute from the key
c. Not BCNF ... e.g. in $C \rightarrow A$, *C* does not contain a key

vi. $A \rightarrow BCD$

[hide answer]

a. Candidate keys: *A*
b. 3NF ... all left hand sides are superkeys
c. BCNF ... all left hand sides are superkeys

- Ass2Q1 (1)

## Question 1 (8 marks)

1) (2 marks)

$\{E, H\}$ or $\{A, B, H\}$ or $\{B, D, H\}$ or $\{C, D, H\}$

Let $X = \{A, B, C, D, E, G, H\}$,

- $A$ can be removed because $\{B, C, D, E, G, H\}^+ = R$, so $X = \{B, C, D, E, G, H\}$;
- $B$ can be removed because $\{C, D, E, G, H\}^+ = R$, so $X = \{C, D, E, G, H\}$;
- $C$ can be removed because $\{D, E, G, H\}^+ = R$, so $X = \{D, E, G, H\}$;
- $D$ can be removed because $\{E, G, H\}^+ = R$, so $X = \{E, G, H\}$;
- $E$ cannot be removed because $\{G, H\}^+ = \{G, H\} \neq R$;
- $G$ can be removed because $\{E, H\}^+ = R$, so $X = \{E, H\}$;
- $H$ cannot be removed because $\{E\}^+ = \{A, B, C, D, E, G\} \neq R$.

Thus, $\{E, H\}$ is a candidate key for $R$.

## 3.4 无损连接分解 lossless-joint

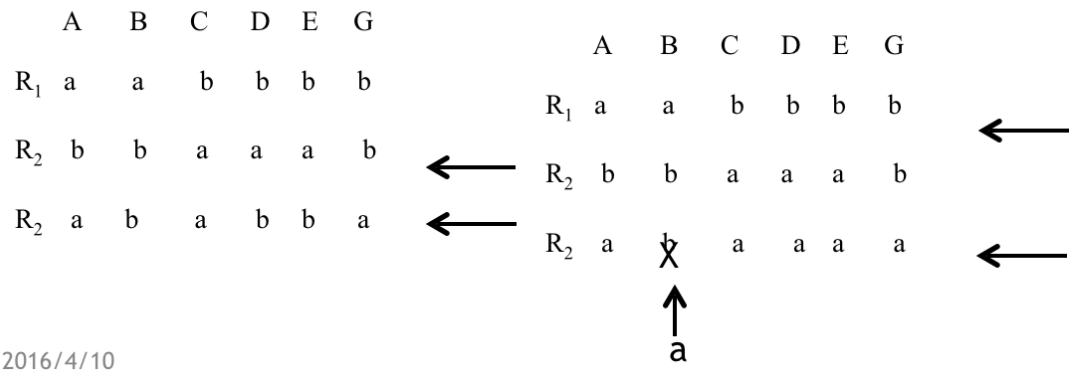
### 3.4.1 check whether lossless-joint

- 画图

*Example 5:* $R = (A,B,C,D,E,\ G)$,

$F = \{AB \rightarrow G,\ C \rightarrow DE,\ A \rightarrow B,\}$.

Let $R_1 = (A,B,)$, $R_2 = (C,\ D,\ E)$ and $R_3 = (A,\ C,G)$.

- 




## 3.4.2 lossless decomposition into 3NF

- 我的解法是这样。。
- step 1 找candidate key
- step 2 将candidate key 作为R1
- step 3 分解candidate key中的单个key，将其各自的关系作为Rn
- step 4 如果Rn中有违反3NF传递，再继续分解

## 3.4.3 lossless decomposition into BCNF（答案不唯一）

- 考题ass2q1(4)
  - F ={AB→ CD,E→ D,ABC→ DE,E→ AB,D→ AG,ACD→ BE }
  - Candicate Key ={E,H}or{C,D,H}or{A,B,H} ,**superkey需要包含candidate key**

    Consider $AB \rightarrow CD$, $AB$ is not a superkey, split $R$ into $R_1\{A,B,C,D\}$ and $R_2\{A,B,E,G,H\}$

    Consider $D \rightarrow A$ in $R_1\{A,B,C,D\}$, $D$ is not a superkey, split $R_1$into $\boldsymbol{R_{11}\{A,D\}}$ and $\boldsymbol{R_{12}\{B,C,D\}}$
  - Consider $E \rightarrow AB$, $E$ is not a superkey, split $R_2$ into $\boldsymbol{R_2\{A,B,E\}}$ and $R_3\{E,G,H\}$

    Consider $E \rightarrow G$, $E$ is not a superkey, split $R_3$ into $\boldsymbol{R_{31}\{E,G\}}$and $\boldsymbol{R_{32}\{E,H\}}$

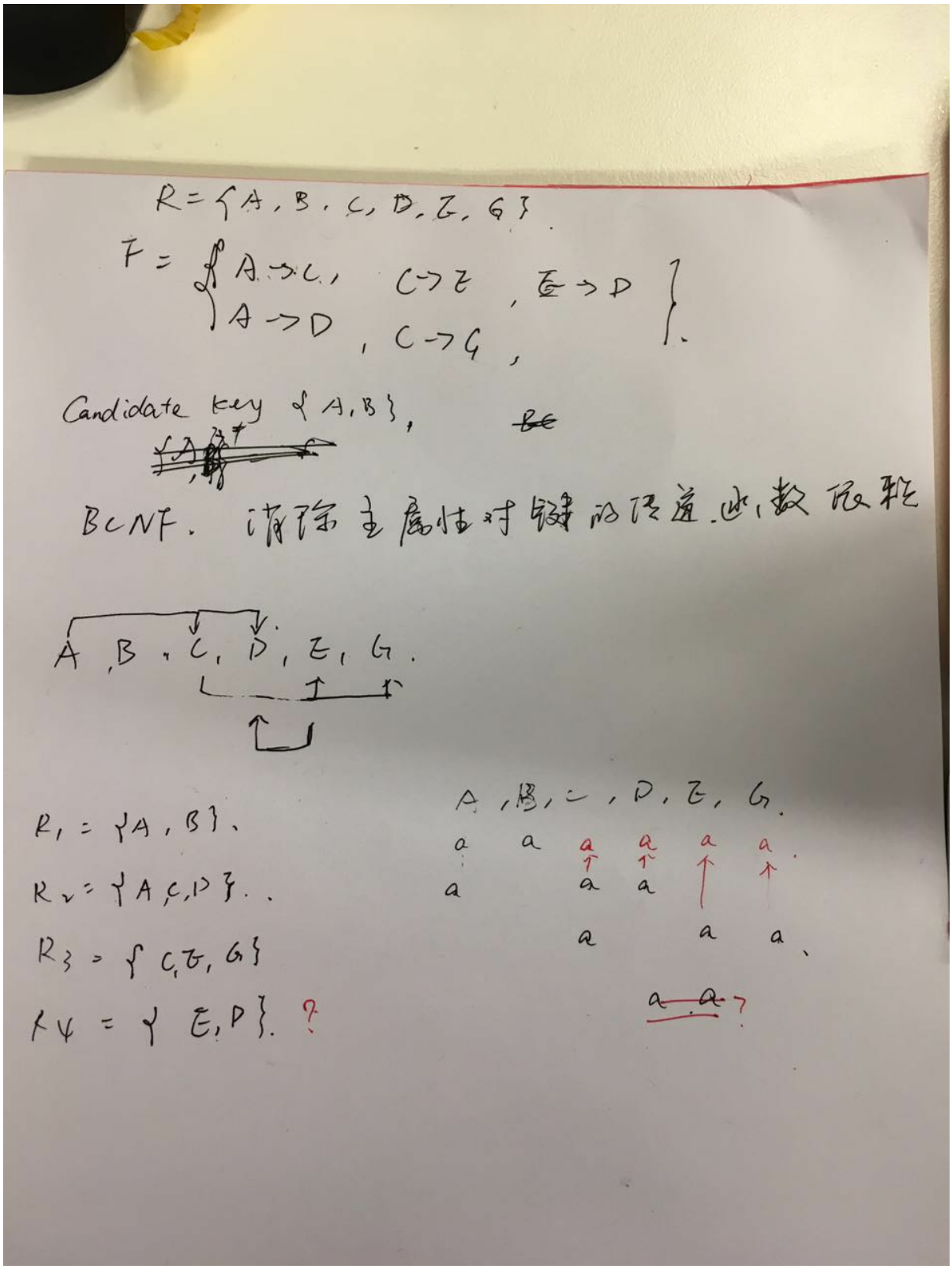
    One of the possible lossless-join decompositions to BCNF is: $R_{11}, R_{12}, R_2, R_{31}, R_{32}$
- 考题 PPT5.3

- 这个PPT的解法很清奇
  - $F = \{A \rightarrow C, A \rightarrow D, C \rightarrow E, E \rightarrow D, C \rightarrow G\},$

    $R1 = (C, D, E, G), R2 = (A, B, C, D)$

    $R11 = (C, E, G), R12 = (E, D)$ due $E \rightarrow D$

    $R21 = (A, B, C), R22 = (C, D)$ because of $C \rightarrow D$
  - R1 ∩ R2 = {C,D} 不是primary key
- 如果照以上面的解法

- 


## 3.5 保持函数依赖分解 dependency-preserving

..

### 3.5.1 check whether dependency—preserving

- F = { A → BC, D → EG, M → A },
  R = (A, B, C, D, E, G, M, A),
  Given R1= ( A, B, C, M) and R2 = (C, D, E, G),
  - ∵ F1 = { A → BC, M → A}, F2 = {D → EG}
  - ∵ F = F1 U F2.
  - ∴ dependency preserving
- 反例：F = { A → BC, D → EG, M → A , M → D },
  R = (A, B, C, D, E, G, M, A),
  Given R1= ( A, B, C, M) and R2 = (C, D, E, G),
  - ∵ F1 = { A → BC, M → A}, F2 = {D → EG}
  - ∵ $M^{+}|_{F1UF2}$ ={M, A, B, C}
  - M→ D is not inferred by F1 U F2.
  - ∴ not dependency preserving

### 3.5.2 dependency—preserving decomposition into 3NF

### 3.5.3 dependency—preserving decomposition into BCNF

## 3.6 dependency—preserving & lossless decomposition 其实 3.4/3.5 都可以用以下解法

https://wenku.baidu.com/view/31154cc09ec3d5bbfd0a7412.html?from=search

Fm

**Step 1 Reduce Right Side:**

$F' = \{AB \to C, AB \to D, E \to D,\ ABC \to D,\ ABC \to E, E \to A, E \to B, D \to A, D \to G, ACD \to B, ACD \to E\}$

**Step 2 Reduce Left Side:**

For $ABC \to D$, $\{AB\}^+ = \{A, B, C, D, E, G\}$. Thus $AB \to D$ is inferred by $F'$. Hence, $ABC \to D$ is replaced by $AB \to D$.

Similarly, we can replace $ABC \to E$ with $AB \to E$, $ACD \to B$ with $CD \to B$, $ACD \to E$ with $CD \to E$.

**Step 3 Reduce Redundancy:**

$\{E\}^+|_{F'-\{E \to D\}} = \{A, B, C, D, E, G\}$, so $E \to D$ is redundant. Thus, we remove it from $F'$.

Similarly, we can remove $AB \to E$ and $CD \to B$.

Thus, $F_m = \{AB \to C, D \to A, D \to G, E \to B, AB \to D, E \to A, CD \to E\}$

This is a sample solution.

dependency—preserving & lossless decomposition

- For $F_m = \{AB \to C, D \to A, D \to G, E \to B, AB \to D, E \to A, CD \to E\}$:

  From $AB \to C, AB \to D$, derive $R_1\{A, B, C, D\}$

  From $D \to A, D \to G$, derive $R_2\{A, D, G\}$

  From $E \to B, E \to A$, derive $R_3\{A, B, E\}$

  From $CD \to E$, derive $R_4\{C, D, E\}$

  None of the relation schemas contains a key of $R$, add one relation schema $R_5\{E, H\}$

# Q4 Transections & Indexes

## 4.1 概念*

- transection -> Serial schedule (unefficient) -> Serializable schedule (Concurrent,BUT confict ) -> lock (dead lock) ->

# Q4 Transections & Indexes

## 4.1 概念（没时间可以不看）

### 4.1.1 Transaction（2-5）, Concurrency（6-7）, Recovery（8—）

- 事务处理: 某些工作的逻辑单元需要多种数据库操作, 用于管理这些单元的技术称作事务处理
- 并发控制: 用于确保多个并发事务不会互相干扰的技术
- 还原机制: 在事故发生后将数据还原到某个特定状态的技术
- 定义: 访问并可能更新数据库中各种数据项的一个程序执行单元(unit) 事务为数据库操作提供了一个从失败中**恢复**到正常状态的方法，同时提供了数据库即使在 异常状态下仍能保持一致性的方法
    - 当多个应用程序在并发访问数据库时，可以在这些应用程序之间提供一个隔离方法，以防止彼此的操作互相干扰
    - 某个事务被提交给DBMS时,DMBS需要确保该事务中的**所有操作都成功完成**且其结果被**永久保存**在数据库中，如果事务中有的操作没有成功完成，则事务中的所有操作都需要被回滚，回到事务执行前的状态**(要么全执行，要么全都不执行)**;同时，该事务对数据库或者 其他事务的执行无影响，所有的事务都**独立运行**
    - ![6.png-200.1kB][210]

### 4.1.2 transection 造成的影响

- read：
- write
- abort（rollback）
- commit

### 4.1.3 schedules

- Serial schedules 串行

Serial execution: **T1** then **T2** or **T2** then **T1**

```
T1: R(X) W(X) R(Y) W(Y)
T2:                    R(X) W(X)
```

or

```
T1:              R(X) W(X) R(Y) W(Y)
T2: R(X) W(X)
```

Serial execution guarantees a consistent final state if

- the initial state of the database is consistent
- **T1** and **T2** are consistency-preserving

- Concurrent shedule 并行

Concurrent schedules interleave **T1**,**T2**,... operations

Some concurrent schedules are ok, e.g.

```
T1: R(X) W(X)       R(Y)       W(Y)
T2:           R(X)       W(X)
```

Other concurrent schedules cause anomalies, e.g.

```
T1: R(X)       W(X)       R(Y) W(Y)
T2:      R(X)       W(X)
```

Want the system to ensure that only valid schedules occur.

4.1.4 串行化 & 可串行性

- Serializable schedule:

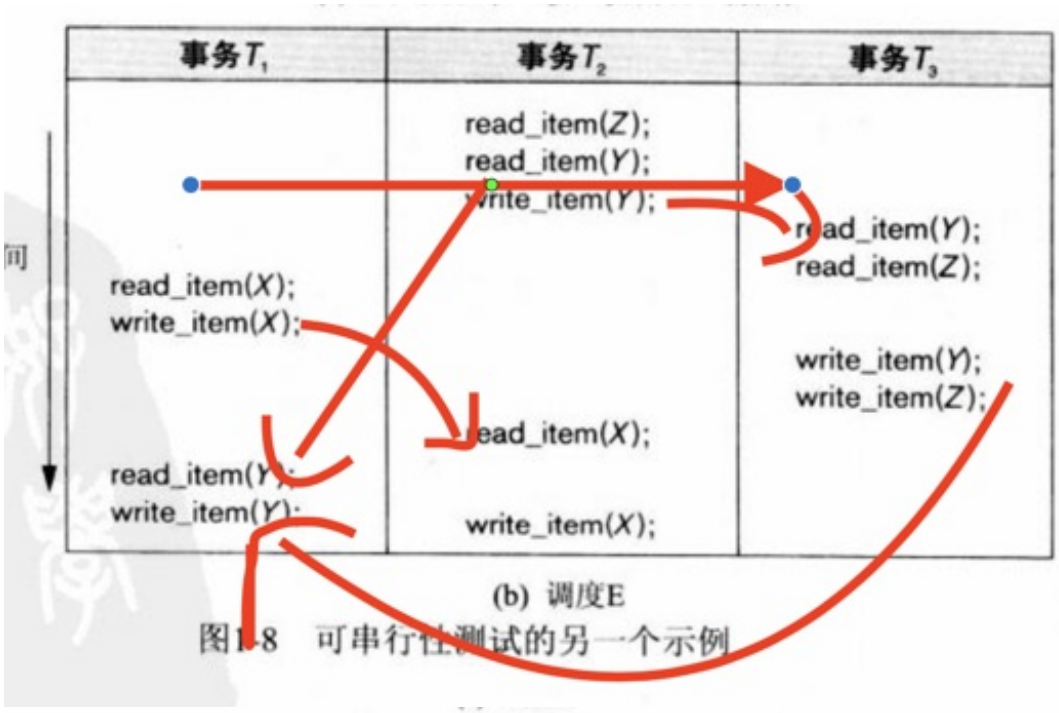
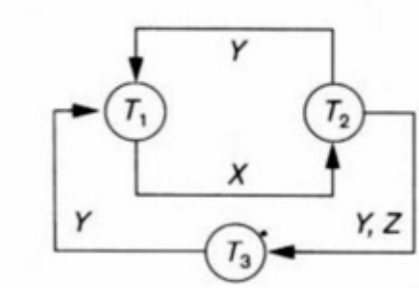
    - 并行concurrent schedule for T1.. Tn with final state S
    - S is also a final state of one of the possible serial schedules for T1..Tn
- 实现串行化化 schedule equivalence
    - conflict serializibility: 读写操作需要保证正确的顺序
    - view serializibility : 读取操作需要保证看到的数据是正确的
- 可串行xing
    - 

图1-8 可串行性测试的另一个示例
    - 


| 等价串行调度 | 原因 |
| --- | --- |
| 没有 | 环 $X(T_1 \rightarrow T_2), Y(T_2 \rightarrow T_1)$<br>环 $X(T_1 \rightarrow T_2), YZ(T_2 \rightarrow T_3), Y(T_3 \rightarrow T_1)$ |

- 串行调度 不同于 可串行化调度
    - 串行调度不允许不同的事物操作交替
    - 所以需要 可串行化调度
    - 但是可串行化调度需要protocal来确保可串行化调度**并发控制**
    - 通常采用二项锁2 phrase lock来yue sh

### 4.1.5 ACID

- Atomic原子性:
    - 某个事务中的操作要么全部成功, 要么全部回退
    - 另一方面在读取数据时, 某个事务对同一数据项多次读取的结果一定是相同的

- Consistency一致性:
  - 

  - 只有合法数据可以被写入数据库
  - 事务需要保证数据库的正确性/完整性/一致性
  - 一致性可通过数据库内部规则或者应用规则进行保证
  - 中间态是inconsistant的, 但是事务发生前后的状态需要一致
- Isolation隔离性
  - 并发事务彼此独立, 不互相干扰
  - 另一方面不允许其他事务看到该事务的中间态(通过另一事务,只能该事务进行前或该事务已经完成提交后数据库产生变化的状态)
  - 一般使用**锁**进行控制(**并行的时候？**)
- Durable持久性
  - 事务提交完成后, 事务处理结果被固化, 即使系统发生故障也可进行还原

### 4.1.6 基于锁的并发控制

#### 4.1.6.1 2PL

- 在读取数据X前, 获取X的共享锁
- 在写入数据X前, 获取X的排它锁
- 事务T对数据A加上共享锁后，则其他事务只能对A再加共享锁，不能加排他锁。获准共享锁的事务只能读数据，不能修改数据
- 事务T对数据A加上排他锁后，则其他事务不能再对A加任任何类型的封锁。获准排他锁的事务既能读数据，又能修改数据。
- **注意如果事务间联通，仅仅有这些规则是无法保证串行性的**
- 锁与性能的关系: 锁减少了并发性, 吞吐量下降
- mvcc
  - 写入操作建立了新版本tuple, 不会导致
  - 读取操作被锁 读取操作读取旧版本的tuple, 不会导致写入操作被锁

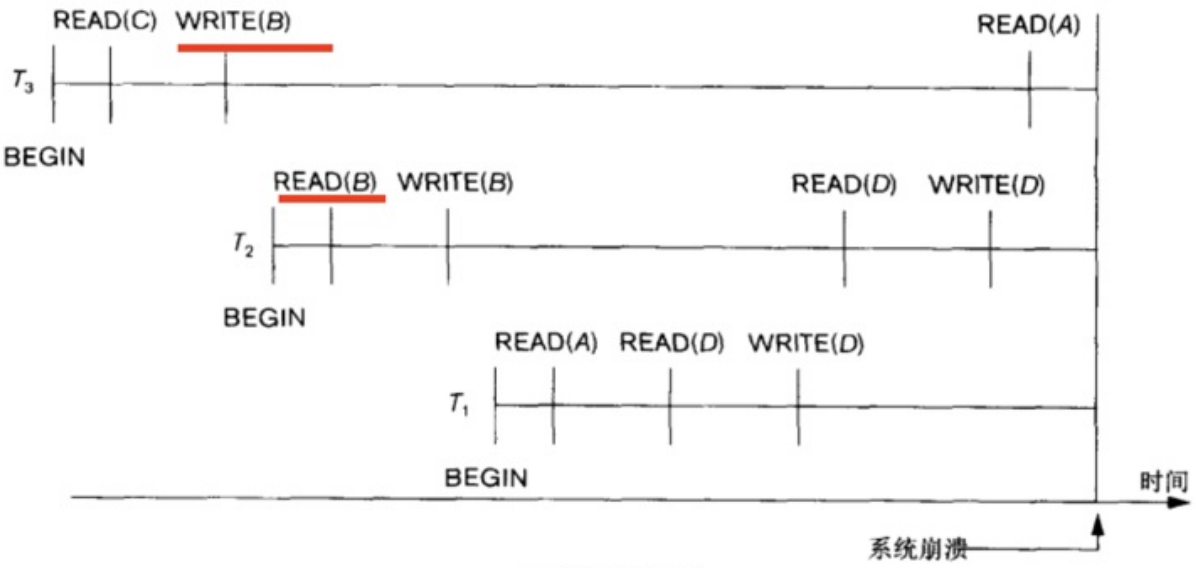
#### 4.1.6.2 时间戳timestamp order

- 解决死锁

- wait-die
- wound-die

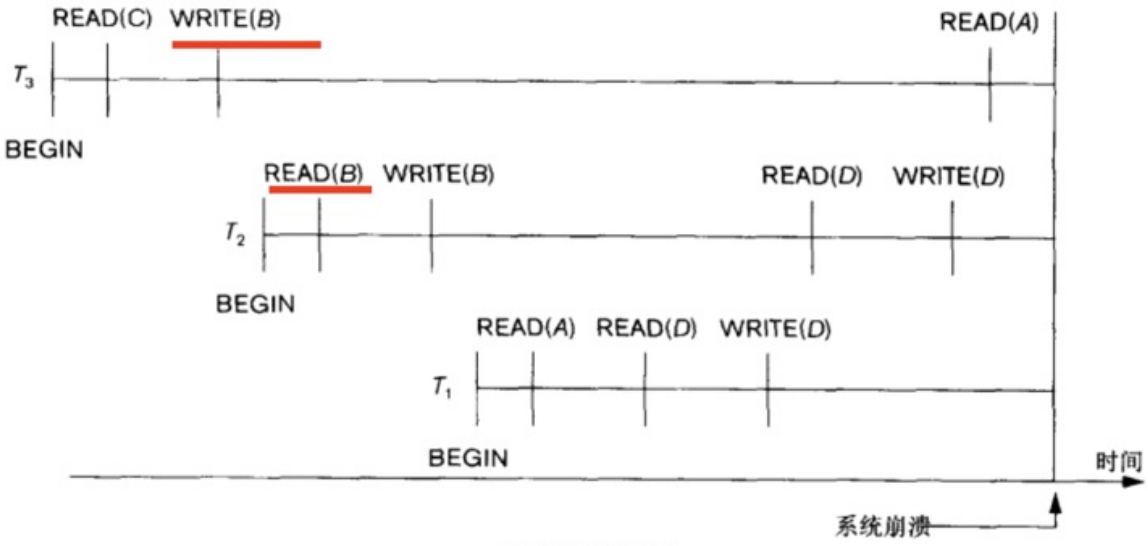
### 4.1.7数据库的隔离级别*

- 从低到高可分为: RU - RC - RR - S
- Read uncommitted(RU):读取未提交的数据(未授权读取)
  - 其他事务已经修改单未commit的数据，这是最低的隔离级别
  - 允许脏读取，但不允许更新丢失
  - 如果一个事务已经开始写数据，则另外一个数据则不允许同时进行写操作，但允许其他事务读此行数据
  - RU可以通过排它锁实现
- Read committed(RC):允许不可重复读取
  - 在一个事务中，对同一个项，前面的读取跟后面的读取结果可能不一样
  - 例如：第一次读取时另一个事务的修改还没有提交，第二次读取时已经提交了
  - RC可以通过瞬间共享锁和排它锁实现
  - 读取数据的事务允许其他事务继续访问该行数据，但是未提交的写事务将会禁止其他事务访问该行
- Repeatable read(RR):可重复读取
  - 在一个事务中，对同一个项，要求前面的读取跟后面的读取结果一样
  - RR可以通过共享锁和排它锁实现
  - 读取数据的事务将会禁止写事务(但允许读事务)，写事务则禁止任何其他事务
- Serializable(S):可序列化
  - 最高执行级别
  - 要求事务序列化执行, 不允许并发执行
  - 如前面所说, **仅仅通过锁无法保证串行化**，必须通过其他机制保证新插入的数据不会被刚执行查询操作的事务访问到

- 

(c) 崩溃前的操作

### 4.1.8 事务恢复 / p41

- defeered updata 延时更新
  - 在提交前崩了，直接redo
- 及时更新
  - 同时用到 redo undo
- 日志
  - read_item 被认为是undo型记录
  - check_point, 崩溃时 无需对check_point 前的commit 的事项做恢复
- 回滚**重点**
  - 

(c) 崩溃前的操作
  - 如果题目中说T3 rollback，那么T2 也要回滚，因为T2读了T3 写的B

## 4.1.9 基于延时更新的恢复（不想考虑这么多 可以照着ppt 的思路做）

- 单用户
  - 因为单线程，对已提交write_item 操作进行redo
  - 没了
- 多用户
  - RDU_M过程（带有检查点）。使用系统维护的两个事务列表：自最后一次检查点以来，所有已提交事务$T$的列表（**提交列表**）和当前活动事务$T'$的列表（**活动列表**）。根据日志记录的顺序，对已提交事务的WRITE操作执行REDO。活动事务和尚未提交的事务立即删除，并必须重新提交。

    图3-3所示是一个可能的执行事务的调度。当在$t_1$时刻执行检查点操作时，事务$T_1$已经提交，但$T_3$和$T_4$尚未提交。系统在$t_2$时刻发生崩溃，此时$T_3$和$T_2$已提交，但$T_4$和$T_5$尚未提交。根据RDU_M方法，无需重做事务$T_1$或其他在最后一次检查点$t_1$前已被提交事务的write_item操作。但必须重做$T_2$和$T_3$的write_item操作，因为这两个事务在最后一次检查点之后才到达提交点。我们在前面已经陈述过，事务提交之前先要强制写日志。事务$T_4$和$T_5$将被忽略，因为使用延迟更新协议时，$T_4$和$T_5$的write_item操作不会对数据库产生任何影响，因此删除或回滚这两个事务。稍后还将用图3-3来说明其他恢复协议。

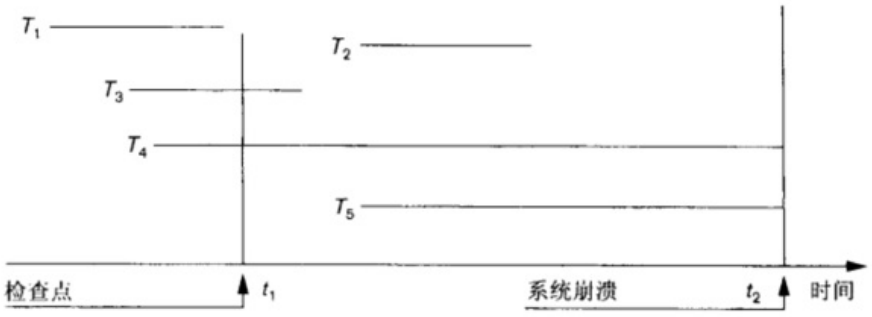



图3-3　多用户环境下的恢复示例

## 4.1.10 基于及时更新的恢复

- 单用户
  - RIU_S过程。

    (1) 使用系统维护的两个事务列表：自最后一个检查点之后的所有已提交事务列表和活动事务列表（最多有一个事务被填入此表，因为是在单用户系统中）。

    (2) 使用下面描述的UNDO过程，对日志中所有活动事务的write_item操作执行取消操作。

    (3) 使用上面描述的REDO过程，根据日志记录的顺序重做所有已提交事务的write_item操作。
- 多用户

RIU_M过程。

(1) 使用系统维护的两个事务列表：自最后一个检查点之后的所有已提交事务列表和活动事务列表。

- (2) 使用UNDO过程取消活动（未提交）事务的所有write_item操作。根据日志记录的相反顺序取消上述操作。

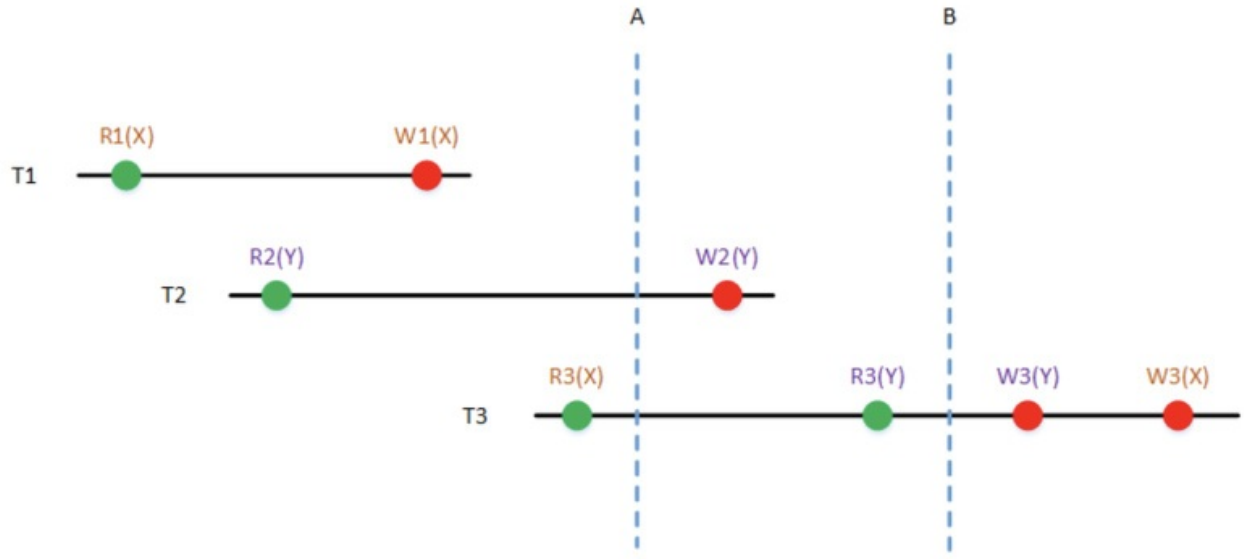
(3) 根据日志记录的顺序重做所有已提交事务的write_item操作。

4.1.n 概念考题 判断，简答6]

4.1.8 概念考题

- 为什么要并发控制
- 解释ACID
- 什么是2PL，它如何保证可串行性
- 时间戳有什么用

## 4.2 冲突调度-redo，undo

- 


1) (2 mark) crash at B

T1, T2: redo

T3: undo

2) (2 mark) crash at B, A is check point

T2: redo

T3: undo

transection 包含多个action

## 4.3 判断死锁

- 判断有没有联通 看4.1.4

## 4.4 index

- https://www.quora.com/What-is-difference-between-primary-index-and-secondary-index-exactly-And-whats-advantage-of-one-over-another
-

| Type of Index | Number of (First-level) Index Entries | Dense or Nondense (Sparse) | Block Anchoring on the Data File |
|---|---|---|---|
| Primary | Number of blocks in data file | Nondense | Yes |
| Clustering | Number of distinct index field values | Nondense | Yes/no[a] |
| Secondary (key) | Number of records in data file | Dense | No |
| Secondary (nonkey) | Number of records[b] or number of distinct index field values[c] | Dense or Nondense | No |

### 4.4.1 primary index

- file 最多只有 一个primary index 指向具体的block

**Example 1.** Suppose that we have an ordered file with $r = 30{,}000$ records stored on a disk with block size $B = 1024$ bytes. File records are of fixed size and are unspanned, with record length $R = 100$ bytes. The blocking factor for the file would be $bfr = \lfloor (B/R) \rfloor = \lfloor (1024/100) \rfloor = 10$ records per block. The number of blocks needed for the file is $b = \lceil (r/bfr) \rceil = \lceil (30000/10) \rceil = 3000$ blocks. A binary search on the data file would need approximately $\lceil \log_2 b \rceil = \lceil (\log_2 3000) \rceil = 12$ block accesses.

- Now suppose that the ordering key field of the file is $V = 9$ bytes long, a block pointer is $P = 6$ bytes long, and we have constructed a primary index for the file. The size of each index entry is $R_i = (9 + 6) = 15$ bytes, so the blocking factor for the index is $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (1024/15) \rfloor = 68$ entries per block. The total number of index entries $r_i$ is equal to the number of blocks in the data file, which is 3000. The number of index blocks is hence $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (3000/68) \rceil = 45$ blocks. To perform a binary search on the index file would need $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 45) \rceil = 6$ block accesses. To search for a record using the index, we need one additional block access to the data file for a total of $6 + 1 = 7$ block accesses—an improvement over binary search on the data file, which required 12 disk block accesses.

### 4.4.2 secondary index

- **Example 2.** Consider the file of Example 1 with $r = 30{,}000$ fixed-length records of size $R = 100$ bytes stored on a disk with block size $B = 1024$ bytes. The file has $b = 3000$ blocks, as calculated in Example 1. Suppose we want to search for a record with a specific value for the secondary key—a nonordering key field of the file that is $V = 9$ bytes long. Without the secondary index, to do a linear search on the file would require $b/2 = 3000/2 = 1500$ block accesses on the average. Suppose that we construct a secondary index on that *nonordering key* field of the file. As in Example 1, a block pointer is $P = 6$ bytes long, so each index entry is $R_i = (9 + 6) = 15$ bytes, and the blocking factor for the index is $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (1024/15) \rfloor = 68$ entries per block. In a dense secondary index such as this, the total number of index entries $r_i$ is equal to the *number of records* in the data file, which is 30,000. The number of blocks needed for the index is hence $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (3000/68) \rceil = 442$ blocks.

  A binary search on this secondary index needs $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 442) \rceil = 9$ block accesses. To search for a record using the index, we need an additional block access to the data file for a total of $9 + 1 = 10$ block accesses—a vast improvement over the 1500 block accesses needed on the average for a linear search, but slightly worse than the 7 block accesses required for the primary index. This difference arose because the primary index was nondense and hence shorter, with only 45 blocks in length.

### 4.4.3 cluster/uncluster index

- cluster index 指向具体的record
- 取值相同的聚合

考题

**Question 3.** (cont) (e) (5 points) When scanning a table to perform a join, the database can either do a full scan of the table or use an index to directly access tuples as needed. Will indexed access always be cheaper than a full scan? Answer yes or no and give a brief explanation supporting your answer.

**No. Accesses to disk blocks are slow (milliseconds) and cost the same whether a single tuple or all of the tuples in the block are processed. If we use an index to access a table, the cost will be proportional to the number of tuples read. That can be significantly more expensive than a sequential scan of the full table if a large percentage of the tuples in the table are retrieved.**

# Q5 空间数据库

TABLE 3.9: A Sample of Operations Listed in the OGIS Standard for SQL [OGIS, 1999]

| | | |
|---|---|---|
| Basic Functions | `SpatialReference()` | Returns the underlying coordinate system of the geometry |
| | `Envelope()` | Returns the minimum orthogonal bounding rectangle of the geometry |
| | `Export()` | Returns the geometry in a different representation |
| | `IsEmpty()` | Returns true if the geometry is a null set |
| | `IsSimple()` | Returns true if the geometry is simple (no self-intersection) |
| | `Boundary()` | Returns the boundary of the geometry |
| Topological/ Set Operators | `Equal` | Returns true if the interior and boundary of the two geometries are spatially equal |
| | `Disjoint` | Returns true if the boundaries and interior do not intersect |
| | `Intersect` | Returns true if the geometries are not disjoint |
| | `Touch` | Returns true if the boundaries of two surfaces intersect but the interiors do not |
| | `Cross` | Returns true if the interior of a surface intersects with a curve |
| | `Within` | Returns true if the interior of the given geometry does not intersect with the exterior of another geometry |
| | `Contains` | Tests if the given geometry contains another given geometry |
| | `Overlap` | Returns true if the interiors of two geometries have nonempty intersection |
| Spatial Analysis | `Distance` | Returns the shortest distance between two geometries |
| | `Buffer` | Returns a geometry that consists of all points whose distance from the given geometry is less than or equal to the specified distance |
| | `ConvexHull` | Returns the smallest convex geometric set enclosing the geometry |
| | `Intersection` | Returns the geometric intersection of two geometries |
| | `Union` | Returns the geometric union of two geometries |
| | `Difference` | Returns the portion of a geometry that does not intersect with another given geometry |
| | `SymmDiff` | Returns the portions of two geometries that do not intersect with each other |

- 
- file 最多只有 一个primary index 指向具体的block
- cluster index 指向具体的record