Database System

My Notes for 23T3 COMP9311 Database System

天天

目录

Week1.1 Subject Intro, Intro to DB

PART

I

Section 1

Two Types of Data

- Unstructured Data
- Structured Data
    1. Stored with a rigid and strict schema
    2. Can be stored into relational database

Week1.2 Conceptual DB Design (ER)

PART

II

Kinds of data model:

- conceptual: abstract, high-level data model. e.g. ER, ODL
- logical: concrete, for implementation. e.g. relational, object-oriented
- physical: internal file storage

Section 2

Conceptual Design

Subsection 2.1

Entity-Relationship Model

Definition 1 chen's ER model has Two major concepts:

- Entity: collection of attributes describing object of interest
- Relationship: an association among several entities

A third components unofficially:

- Attribute: property of an entity

*NOTE: The ER model is not a standard, so many variations exist.

Subsection 2.2

Some terms in ER model

Entity entity represent the real-world object(rectangle)

1. In a company: employee, department, city, sample
2. In a university: student, course, instructor

Simple Attributes (or Atomic Attributes) are attributes that are not divisible.

value set(or domain of values) is the set of allowed values for each attribute

- For example:

  Entity = student

  Attributes = Student num, name, address, phone_number, ...

- Studnet A:

  Student num = z12345678

  Name = Bob

  ...

Multi-Valued Attribute is an attribute that can have more than one value.

- has more than one value
- no longer a simple attribute

For example: A model shirts with two colors

Composite Attribute can be divided into smaller subparts, which represent more basic attributes with independen meanings.

Some semantics cannot be captured using atomic attributes

For example: An address can be divided into street, city, state, and zip code. And Street_address can be divided into number, street, and apartment_number.

whether an attribute is composite or not depends on the semantics of the application:

if the composite attribute is referenced only as a whole, there is no need to subdivide it into composite attributes. (If the end-user only interested in the whole address)

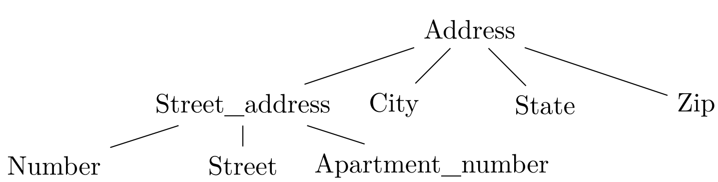

图 **1**. A hierarchy of composite attributes

Derived Attributes some attributes can be computed from other attributes.(they are dynamic and change over time)

For example: age, which can be derived from the birth date.

Entity Type defines a collection of entities that have the same attributes.

- An entity type describes the schema or intension for a set of entities that share the same structure.

- An entity type is represented in ER diagrams as a rectangle box enclosing the entity type name.
- The collection of entities of a particular entity type is gouped into an entity set. Which is also called the extension of the entity type.

  Entity Type Example: Car Entity Schema

  (Registeration(Number, State), Vehicle_id, Make, Model, Year, {Color})

  Car1: ((ABC123, NSW), TK629, Ford Mustang, convertible, 2004, {Red, Blue})

  Car2: ((ABC123, NerYork), WP9872, Nissan Maxima, 4-door, 2005, {Blue})

Week2.1 Relational Data Model

PART

III

Section 3

Structures

In the relational model, everything is described using relations

A relation can be thoughted as a named table.

each column → named aattribute

each row → tuple of relation

The set of allowed values for an attribute is acalled its domain.

PART

IV

Week2.2 Relational Algebra

不高兴写

Week3.1 SQL

PART

V

没学

Week3.2 SQL, PLpgSQL

PART

VI

没学

PART

VII

Week4.2 PLpgSQL

没学

Week5.1 Functional Dependencies

PART VIII

Section 4

Functional Dependencies

Subsection 4.1

How good is a DB design?

- Conceptual Level

  How do users interpret the relation schema and the meaning of their attributes?

  Information Preservation

  – the design correctly capture all attributes, entities and relations.

- Physical Level

  How the tuples in a base relation are stored and updated?

  Minimum Redundancy

  – design minimize redundant storage of the same information and reduce the need for multiple updates.

Subsection 4.2

Devise a Theory for what is Good

We want two things:

- each relation is in good form
- the decomposition is a lossless

Subsection 4.3

Functional Dependencies

Definition 2

Functional Dependency

A functional dependency is a one-way relationship between two attributes, such that at any given time, for each unique value of attribute α, only one value of attribute β is associated with it throughout the relation.

Notation: $\alpha \to \beta$

Functional Dependencies exist to:

- capture semantics of the real-world problem
- describe constraints on the data
- describe relationships between attributes (which is not captured by ER)

SECTION 5

Armstrong's Axioms

Axioms 1

Rule1: Reflexivity

if $\beta \subseteq \alpha$, then $\alpha \to \beta$

e.g. $\{X, Y\} \to \{X\}$

Axioms 2

Rule2: Augmentation

if $\alpha \to \beta$, then $\gamma\alpha \to \gamma\beta$

e.g. $X \to Y \Rightarrow ZX \to ZY$

Axioms 3

Rule3: Transitivity

if $\alpha \to \beta$, and $\beta \to \gamma$ then $\alpha \to \gamma$

e.g. $X \to Y$ and $Y \to Z \Rightarrow X \to Z$

Axioms 4

Rule4: Additivity

if $\alpha \to \beta$ holds and $\alpha \to \gamma$ holds, then $\alpha \to \beta\gamma$ holds

e.g. $X \to Y$ and $X \to Z \Rightarrow X \to YZ$

Axioms 5

Rule5: Projectivity

if $\alpha \to \beta\gamma$ holds, then $\alpha \to \beta$ holds and $\alpha \to \gamma$ holds

e.g. $X \to YZ \Rightarrow X \to Y$ and $X \to Z$

Axioms 6

Rule6: Pseudo-Transitivity

if $\alpha \to \beta$ holds and $\gamma\beta \to \delta$ holds, then $\alpha\gamma \to \delta$ holds

e.g. $X \to Y$ and $YZ \to W \Rightarrow XZ \to W$

Note: Rule4, Rule5, Rule6 can be derived from Rule1, Rule2, Rule3

Subsection 5.1

Inferring Other FDs

Definition 3

Logically Implied

A set of FDs F logically implies a FD $\alpha \to \beta$ if $\alpha \to \beta$ is logically derivable from F.

Notation: $\mathrm{F} \models \alpha \to \beta$

Example

e.g.

$\mathrm{F} = \{\ A \to B,\ B \to C\ \}$

$\mathrm{F} \models A \to C$

Subsection 5.2

Clousure of F

Definition 4

Closure of F

The closure of F, denoted by F^+, is the set of all functional dependencies that can be inferred from F using the rules of inference.

Denotion:

$F^+ = \{\ \alpha \to \beta \mid \mathrm{F} \models \alpha \to \beta\ \}$

Algorithm 1 The Procedure for Computing F^+

$F+ := F$;
repeat
  for each functional dependency $X \to Y$ in $F+$ **do**
    apply **Reflexivity** and **Augmentation** rules on f;
    add the resulting functional dependencies to F^+;
  end for
  for each pair of functional dependencies $f1$ and $f2$ in F^+ **do**
    if $f1$ and $f2$ can be combined using **Transitivity then**
      add the resulting functional dependency to F^+;
    end if
  end for
until no more changes occur;

Week5.2 Normal Forms

PART

IX

SECTION 6

Normal Forms

If a table has data redundancy and is not properly normalized, then it will be difficult to handle and update the database, without facing data loss. It will also eat up extra memory space and Insertion, Update, and Deletion Anomalies are very frequent if the database is not normalized.

Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. Normal forms are used to eliminate or reduce redundancy in database tables.

Definition 5 **First Normal Form (1NF)**

A relation schema R is in 1NF if all the attribute values are **atomic**, and are part of the definition of the relational model.

Atomic: multivalued attributes, composite arrtibutes, and their combinations are disallowed.

Prof	Course	Fac_Dept	Crs_Dept
Smith	353	Comp Sci	Comp Sci
Smith	221	Comp Sci	Decision Sci
Turner	353	Chemistry	Comp Sci
Turner	456	Chemistry	Mathematics

表 1. A relation that is in 1NF

Above table is in 1NF because all the values are atomic. It's not in 2NF, because there are partial dependencies.

Definition 6 **Second Normal Form (2NF)**

2NF Prerequisite

Full functional dependency: In an FD $X \to Y$, Y is fully funcitionally dependent on X if there is no $Z \subset X$ such that $Z \to Y$

Partial functional dependency: In an FD $X \to Y$, Y is partially functionally dependent on X if there is any $Z \subset X$ such that $Z \to Y$

Definition of Second Normal Form

A relation schema R is in second normal form(2NF) if every nonprime attribute A in R is not partially dependent on any key of R.
Note: ”key” here refers to the candidate key.

Primary Attribut: An attribute that is part of a candidate key is called a primary attribute.

Nonprime Attribute: An attribute that is not part of any candidate key is called a nonprime attribute.

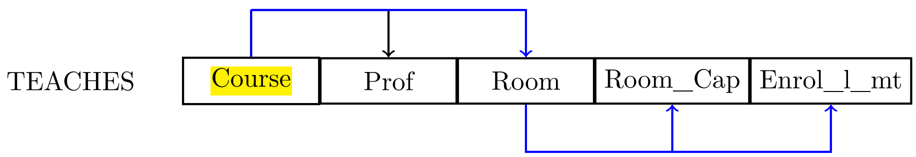

图 **2**. A relation that is in 2NF

We can notice that 2NF allows transitive dependencies. In this case, $Course \to Room \to \{Room\_Cap, Enrol\_l\_mt\}$ is a transitive dependency (a nonprime attribute $\to$ nonprime attribute).

Definition 7 **Third Normal Form (3NF)**

3NF Prerequisite

Transitive dependency: A FD $X \to Y$ is a transitive dependency if there is a Z that is not a subset of any key, such that $X \to Z$ and $Z \to Y$. The attributes of Y are transitively dependent on X.

Definition of Third Normal Form

A relation schema R is in third normal form(3NF) if for all non-trivial FDs of the form

$$X \to A$$

Either:

- X is a superkey of R, or
- A is a prime attribute of R

The 3NF disallows FDs of the form ”Not superkey $\to$ Nonprime”

X	Y
Superkey	Prime
Superkey	Nonprime
Not Superkey	Prime
~~Not Superkey~~	~~Nonprime~~

表 **2**. FDs in 3NF

On the base of 2NF, 3NF disallows a nonprime $\rightarrow$ a nonprime.

Subsection 6.1

Decomposition

Redundancy/Anomalies can be removed from relation designs by decomposing them until they are in a (higher) Normal Form.

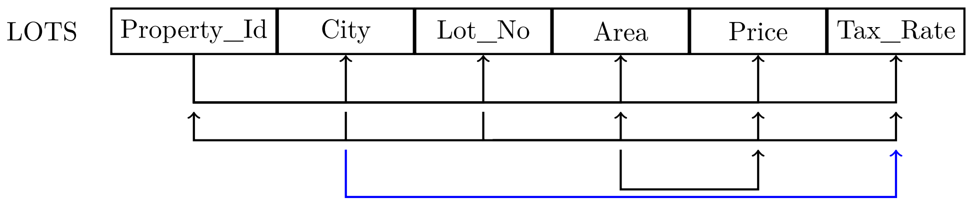

图 **3**. Original Relation LOTS

LOTS is not in 2NF:

Step1 Find all Candidate Keys: {Property_Id}, {City, Lot_No}

Step2 Remove Partial Dependencies:

> Because {City} is part of candidate key {City, Lot_No}, and {City} $\rightarrow$ {Tax_Rate} which is a partial dependency, so LOTS is not in 2NF. So we need to decompose LOTS.

After first decomposition, we get:

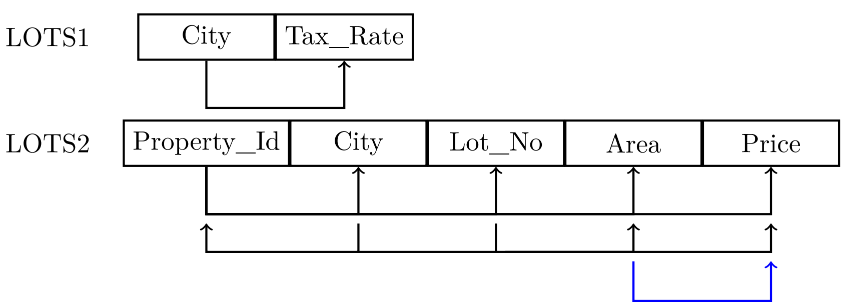

图 **4**. LOTS after first decomposition

Now LOTS meets 2NF, but not 3NF.

Step3 Remove Transitive Dependencies:

> Because {Area} is not a superkey, and {Area} $\rightarrow$ {Price} is a nonprime $\rightarrow$ nonprime dependency. So we need to decompose LOTS2 again.

After second decomposition, we get:
Now LOTS meets 3NF (and also BCNF).

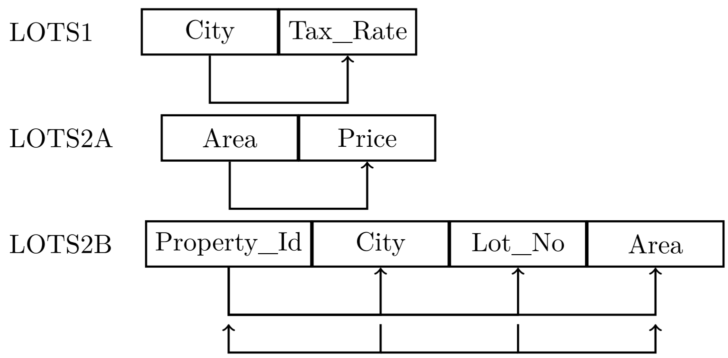

图 **5**. LOTS after second decomposition

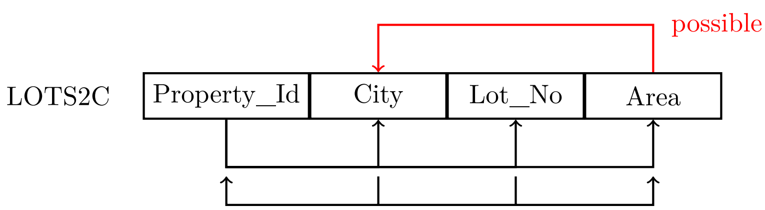

图 **6**. LOTS2C is not in BCNF

Imagine that there could be a new FD: {Area} $\rightarrow$ {City}, then LOTS2C is not in BCNF.

Definition 8

Boyce-Codd Normal Form (BCNF)

A relation schema R is in Boyce-Codd Normal Form (BCNF) if whenever $X \rightarrow A$ holds and $X \rightarrow A$ is a non-trivial, X is a superkey.

X	Y
Superkey	Prime
Superkey	Nonprime
~~Not Superkey~~	~~Prime~~
~~Not Superkey~~	~~Nonprime~~

表 **3**. FDs in BCNF

We can decompose LOTS2C to meet BCNF:

Step4 Remove FDs of the form ”Not Superkey → Prime”:

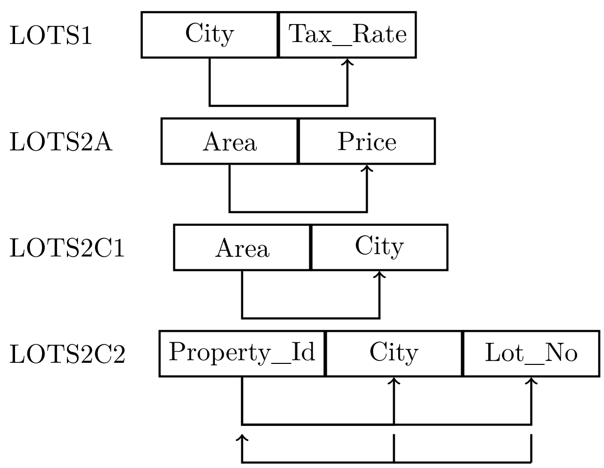

图 **7**. LOTS is in BCNF

Subsection 6.2

Comparisons

BCNF implies 3NF, which implies 2NF, which implies 1NF.

Property	**3NF**	**BCNF**
Elimination of redundancy due to FDs	Most	Yes
Lossless Join	Yes	Yes
Dependency Preservation	Yes	Maybe

表 **4**. Comparisons of 3NF and BCNF

Week7.1 Relational Databse Design

PART

X

Definition 9

Decomposition

A decomposition of a relation schema R is a set of relation schema $\{R_1, ..., R_n\}$ such that $R_i \subseteq R$ for each i, and $U_{i=1}^{n} R_i = R$
Note: This is called **attribute preservation** condition of decomposition.

A good decomposition should satisfy the following properties:

- the **dependency preservation** property
- the **nonadditive(or lossless)** join property

Section 7

Dependency Preservation

Definition 10

Dependency Preservation

A decomposition D=$\{R_1, ..., R_n\}$ is dependency preserving wrt a set F of FDs if:

$$(F_1 \cup ... \cup F_n)^+ = F^+$$

where F_i means the **projection** of F onto R_i.

Definition 11

Projection of F

The projection of F on R_i, denoted by $\pi_{R_i}(F)$ where R_i is a subset of R, is the set of dependencies $X \to Y$ in F^+ such that the attributes in $X \cup Y$ are all contained in R_i.
Note: To simplify notations, we denote the projection of F on R_i by F_i.

In simple English: F_i is the subset of dependencies in F^+ that include only attributes in R_i.
Here is a simple example of function projection:

Example **Example1**

$$R = (A, B, C, D, E, G, M)$$
$$F = A \to BC, D \to EG, M \to A$$
Decomposition into R_1 and R_2
$$R_1 = (A, B, C, M),$$
$$Projection of R_1(F) = \{A \to BC, M \to A\}$$
$$R_2 = (C, D, E, G),$$
$$Projection of R_2(F) = \{D \to EG\}$$

We only checked if $F_1 \cup F_2$ is the same as F, this **is not always sufficient**.
Here is another example:

Example **Example2**

$$R = (A, B, C, D, E, G, M)$$
$$F = A \to BC, D \to EG, M \to A, M \to D$$
Decomposition into R_1 and R_2
$$R_1 = (A, B, C, M),$$
$$Projection of R_1(F) = \{A \to BC, M \to A\}$$
$$R_2 = (A, D, E, G),$$
$$Projection of R_2(F) = \{D \to EG\}$$

In this exmaple, we cannot **get FD $M \to D$ from** $(F_1 \cup F_2)^+$. So this decomposition is not dependency preserving.
Here is another example:

Example **Example3**

$$R = (A, B, C, D, E, G, M)$$
$$F = A \to BC, D \to EG, M \to A, M \to D, A \to D$$
Decomposition into R_1 and R_2
$$R_1 = (A, B, C, M),$$
$$Projection of R_1(F) = \{A \to BC, M \to A\}$$
$$R_2 = (A, D, E, G),$$
$$Projection of R_2(F) = \{D \to EG, A \to D\}$$

In this example, we can get FD $M \to D$ from $(F_1 \cup F_2)^+$: $M \to A \to D$, so this decomposition is dependency preserving.

Section 8

Lossless Join

Definition 12 **Lossless Join**

Formally, a decomposition D={R_1, ..., R_n} of R has the lossless join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds, where * is the **NATURAL JOIN** of all the relations in D:

$$*(\pi_{R_1}(r), ...\pi_{R_m}(r)) = r$$

Subsection 8.1

Test Lossless Join

Theorem 1 **Test lossless join for binary decomposition**

The decomposition {R_1, R_2} of R is lossless if the common attributes of $R_1 \cap R_2$ form a **superkey** for either R_1 or R_2.

For example:

Example **Example4**

$R = (A, B, C)$

$F = A \to B$

Decomposition into R_1 and R_2

$R_1 = (A, B),$

$R_2 = (A, C),$

Common attributes of R_1 and R_2

$R_1 \cap R_2 = (A)$

And A is a superkey of R_1

So, this decomposition is a lossless

Algorithm 2 Test Lossless Join Property

INPUT: R of n attributes has been decomposed into m relations: R_1, ..., R_m
OUTPUT: Whether the decomposition is lossless join

Create a **matrix** S of size $m \times n$;
for each relation R_i **do**
  for each attribute A_j of R **do**
    if $A_i \in R_j$ **then**
      $s_{i,j} = a$;
    else
      $s_{i,j} = b$;
    end if
  end for
end for
repeat
  For each $X \to Y$, choose the rows where the elements corresponding to X take the value a;
  In those chosen rows(must be at least two rows), the elements corresponding to Y also take the value a if one of the chosen rows tabke the value a on Y;
until no more changes occur or one row is all a

Example

Example of Test Lossless Join Property

$R = (A, B, C, D, E, G)$

$F = C \to DE, A \to B, AB \to G$

Decomposition into R_1, R_2 and R_3

$R_1 = (A, B)$,

$R_2 = (C, D, E)$,

$R_3 = (A, C, G)$,

	A	B	C	D	E	G
R_1	a	a	b	b	b	b
R_2	b	b	a	a	a	b
R_3	a	b	a	b	b	a

Step1 Check All FDs:

- For $C \to DE$, choose the rows where the elements corresponding to C take the value a: R_2, R_3, and DE in R_3 are b, so we change DE in R_3 to a;

	A	B	C	D	E	G
R_1	a	a	b	b	b	b
R_2	b	b	a	a	a	b
R_3	a	b	a	a	a	a

- For $A \to B$, rows where $A = a$ and $B = a$: R_1, rows where $A = a$ and $B = b$: R_3, so change B in R_3 to a;

	A	B	C	D	E	G
R_1	a	a	b	b	b	b
R_2	b	b	a	a	a	b
R_3	a	a	a	a	a	a

Step2 Check if one row is all a: R_3 is all a, so this decomposition is lossless join.

SUBSECTION 8.2

Lossless Decomposition into BCNF

Algorithm 3 Lossless Decomposition into BCNF

INPUT: D := $\{R_1, R_2, ..., R_n\}$ is a decomposition of R
OUTPUT: A lossless decomposition of R into BCNF

while there exists a $R_i \in D$ and R_i is not in BCNF **do**
  find a FD $X \to Y$ in R_i that violates BCNF;
  replace R_i in D by $(R_i - Y)$ and $(X \cup Y)$;
end while

Example

Step by Step Example of Lossless Decomposition into BCNF

$F = A \to B, A \to C, A \to D, C \to E, E \to D, C \to G$

$R_1 = \{C, D, E, G\}$,

$R_2 = \{A, B, C, D\}$

Step1 Check if R_1 is in BCNF, its not because: C is a superkey and $C \to E \to D$ is a transitive dependency. So we replace R_1 by $(R_1 - D)$ and $(E \cup D)$:

$R_{11} = \{C, E, G\}$,

$R_{12} = \{E, D\}$

Now R_1 is in BCNF.

Step2 Check if R_2 is in BCNF, its not because: A is a superkey and $C \to D$ which can be inferred from $C \to E$ and $E \to D$. So we replace R_2 by $(R_2 - D)$ and

$(C \cup D)$:

$R_{21} = \{A, B, C\}$,

$R_{22} = \{C, D\}$

SUBSECTION 8.3

Lossless and dependency-preserving decomposition into 3NF

Definition 13 **Equivalence**

Two sets of FDs F_1 and F_2 are equivalent if $F_1^+ = F_2^+$.

Definition 14 **Cover**

A set of functional dependencies F is said to cover another set of functional dependencies E if every FD in E is also in F^+; that is, if every dependency in E can be inferred from F; alternatively, we can say that E is covered by F.

Therefore, equivalence means that every FD in E can be inferred from F, and every FD in F can be inferred from E.
E is equivalent to F if and only if E covers F and F covers E.

SUBSECTION 8.4

Minimal Cover

Definition 15 **Minimal Cover**

A minimnal cover F_{min} of a set of functional dependencies E is a minimal set of dependencies(in the standard canonical form and without redundancy) that is **equivalent** to E.

Theorem 2 **A set of F of FDs is minimal if**

- every FD $X \to Y$ in F is simple: Y consists of a single attribute
- every FD $X \to A$ in F is left-reduced: there is no proper subset $Y \subset X$ such that $X \to A$ can be replaced with $Y \to A$.
- No FD in F can be removed: that is, there is no FD $X \to A$ in such that $(F - \{X \to A\})^+ = F^+$.

Example **Computing a Minimal Cover**

Algorithm 4 Algorithm for Minimal Cover

INPUT: a set F of functional dependencies
OUTPUT: a minimal cover F_{min} of F

STEP1 Reduce right side
for each FD $X \to Y$ where Y =$\{A_1, A_2, ..., A_k\}$ **do**
  Replace $X \to Y$ by a set of FDs $X \to A_i$ for $\{1 \leq i \leq k\}$;
end for

STEP2 Reduce left side
for each FD $X \to A \in F$ where X =$\{A_i : 1 \leq i \leq k\}$ **do**
  FOR i = 1 to k, replace X with $X - \{A_i\}$ if $A \in (X - \{A_i\})^+$;
end for

STEP3 Remove redundant FDs
for each FD $X \to A \in F$ **do**
  Remove $X \to A$ from F if $A \in X^+$ with respect to $F - \{X \to \{A\}\}$
end for

$R = A, B, C, D, E, G$

$F = A \to BCD, B \to CDE, AC \to E$

Step 1 Reduce right side

$$\begin{array}{lll} A \to B & B \to C & AC \to E \\ A \to C & B \to D & \\ A \to D & B \to E & \end{array}$$

Step 2 Reduce left side

Because $A \to C$, So $A \to AC \to E$

$$\begin{array}{lll} A \to B & B \to C & A \to E \\ A \to C & B \to D & \\ A \to D & B \to E & \end{array}$$

Step 3 Remove redundant FDs

Because $A \to B$ and $B^+ = \{B, C, D, E\}$

$$\begin{array}{ll} A \to B & B \to C \\ & B \to D \\ & B \to E \end{array}$$

Conclusion $F_{min} = \{A \to B, B \to C, B \to D, B \to E\}$

Algorithm 5 3NF Decomposition Algorithm

INPUT: D := $\{R_1, R_2, ..., R_n\}$ is a decomposition of R
OUTPUT: A decomposition of R into 3NF

Find a minimal cover G for F.
for each left-hand-side X of a FD that appears in G **do**
  Create a relation schema in D with attributes $\{X \cup \{A_1\} \cup \{A_2\} ... \cup \{A_k\}\}$;
  Where $A_1, A_2, ..., A_k$ are all the attributes that appear on the right-hand-side of FDs in G with X as the left-hand-side;
end for
if none of the relation schemas in D contains a key of R **then**
  Add a relation schema to D with attributes that form a key of R;
end if
Eliminate redundant relations:
A relation R is redundant if R is a projection(subset) of another relation in D.

Example

Example of 3NF Decomposition Algorithm

$R = (A, B, C, D, E, G)$

$F_{min} = \{A \to B, B \to C, B \to D, B \to E\}$

Candidate key:(A, G)

Step1 Minimal Cover

$$F_c = \{A \to B, B \to C, B \to D, B \to E\}$$

Step2 For each left-hand-side, we build a new relation:

$$\begin{cases} R_1 = \{AB\} \\ R_2 = \{BCDE\} \end{cases}$$

Step3 Because there is not a key in any relation, so we add a new relation:

$$R_3 = \{AG\}$$

Conclusion $D = \{R_1, R_2, R_3\}$

Week7.2 Disk, File, Index

PART

XI

巴巴巴巴课件上一堆搞的跟文科一样，懒得写了捏

Section 9

Buffer Replacement Policies

There are three main policies:

- Least Recently Used(LRU): Replace the page that has been unused for the longest time.
- Most Recently Used(MRU): Replace the page that has been used most recently.
- First In First Out(FIFO): Replace the page that has been in the buffer for the longest time.

Subsection 9.1

Example of three policies

巴巴巴巴概念一大堆，不直接上例子有啥用（
Consider the following query:

$$P1, P2, P3, P4, P3, P1, P5, P2, P3, P6, P7, P5, P2, P1.$$

9.1.1 FIFO

	P1	P2	P3	P4	P3	P1	P5	P2	P3	P6	P7	P5	P2	P1
Buffer 1	P1	P1	P1	P1	P1	P1	P5	P5	P5	P5	P5	P5	P5	P1
Buffer 2		P2	P2	P2	P2	P2	P2	P2	P2	P6	P6	P6	P6	P6
Buffer 3			P3	P3	P3	P3	P3	P3	P3	P3	P7	P7	P7	P7
Buffer 4				P4	P4	P4	P4	P4	P4	P4	P4	P4	P2	P2
hit	✓	✓	✓	✓			✓			✓	✓		✓	✓

图 8. process of blocks in FIFO

total of hits: 9

9.1.2 MRU

	P1	P2	P3	P4	P3	P1	P5	P2	P3	P6	P7	P5	P2	P1
Buffer 1	P1	P1	P1	P1	P1	P1	P5	P5	P5	P5	P5	P5	P5	P5
Buffer 2		P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P1
Buffer 3			P3	P3	P3	P3	P3	P3	P3	P6	P7	P7	P7	P7
Buffer 4				P4	P4	P4	P4	P4	P4	P4	P4	P4	P4	P4
hit	✓	✓	✓	✓			✓			✓	✓			✓

图 **9**. process of blocks in LRU

total of hits: 8

9.1.3 LRU

	P1	P2	P3	P4	P3	P1	P5	P2	P3	P6	P7	P5	P2	P1
Buffer 1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P6	P6	P6	P6	P1
Buffer 2		P2	P2	P2	P2	P2	P5	P5	P5	P5	P7	P7	P7	P7
Buffer 3			P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P2	P2
Buffer 4				P4	P4	P4	P4	P2	P2	P2	P2	P5	P5	P5
hit	✓	✓	✓	✓			✓	✓		✓	✓	✓	✓	✓

图 **10**. process of blocks in LRU

total of hits: 11

Week8 Transaction Management

PART XII

Section 10

Transaction

Definition 16 **Transaction**

A transaction is a unit of program execution that accesses and possibly updates various data items.

Subsection 10.1

Two main issues

- Failures of various kinds, such as hardware failures and system crashes.
- Concurrent execution of multiple transactions.

Subsection 10.2

ACID Properties

Definition 17 **ACID** To preserve the integrity of data, the database system must ensure the **ACID** property.

- **Atomicity**: The changes caused by the transaction are **atomic**, that is, either all the changes are performed or none of them is.
- **Consistency**: Every transaction sees a consistent database.
- **Isolation**: Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transcations.
- **Durability**: After a transaction completes sucessfully, the changes it has made to the database persist, even if there are system failures.

Subsection 10.3

Transaction States

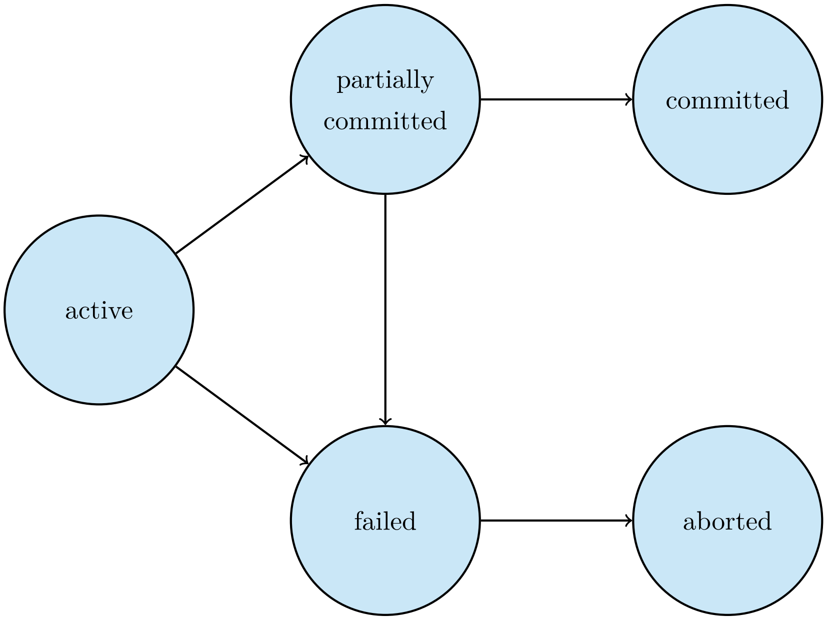

图 **11**. Transaction States

Definition 18

Transaction States
Active: the initial state; the transaction stays in this state while it is executing
Partially committed - after the final statement has been executed.
Failed - after the discovery that normal execution can no longer proceed. **Aborted** - after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:

- Restart the transaction
- kill the transaction

Committed - after successful completion.

Section 11

Serializability

Definition 19

Serializability
A (possibly concurrent) schedule S of n transactions is **serializable** if:
It is equivalent to some serial schedule of the same n transaction.

Subsection 11.1

A simplified View of Transactions

In the following discussion, we pay attention only to the **read** and **write** operations of a transaction.(Do not consider the other operations like computation, output, etc.)

Subsection 11.2

Conflict Instructions

Instructions I_i and I_j of **different transactions** T_i and T_j respectively, Conflict if and only if there exists some item Q accessed by both I_i and I_j, and at least one of these instructions wrote Q.

- I_i = **read(Q)**, I_j = **write(Q)**. I_i and I_j don't conflict.
- one of I_i and I_j is **write(Q)**, I_i and I_j conflict.

In Conclusion, Two operations O_1 and O_2 are conflicting if:

- They are in different transactions
- They access the same data item
- At least one of them must be a write

Section 12

Conflict

Subsection 12.1

Conflict Equivalence

Definition 20 **Conflict Equivalence**

Two schedules S and S' are conflict equivalent if the following conditions hold:

- The two schedules involve the same actions of the same transactions.
- Every pair of conflicting actions is ordered the same way.

Subsection 12.2

Conflict Serializability

Definition 21 **Conflict Serializability**

A schedule S is conflict serializable if it is (conflict) equivalent to some serial schedule S'. (By reordering non-conflicting operations)

T1	T2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

表 **5**. Schedule S

T1	T2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

表 **6**. Schedule S'

12.2.1 Example of Conflict Serializability

In Schedule S, we can change the actions of t4, t5 lines with t6, t7, then we get Schedule S' which is a serial schedule. So S and S' are **conflict equivalent**. So S is **conflict serializable**.

12.2.2 Testing Conflict Serializability

There is a simple algorithm to test whether a schedule is conflict serializable which only considers the **read** and **write** operations of a transaction.

Algorithm 6 Testing Conflict Serializability F^+

STEP1 Construct a precedence graph
for each conflict pair from T_m and T_n about data item Q **do**
  add an edge from T_i to T_j with label;
end for
STEP2 Check if the graph is cyclic:
if the graph is cyclic **then**
  the schedule is not conflict serializable(**non-serializable**);
else
  the schedule is **conflict serializable**;
end if
(**Note Cyclic**: G is cyclic if G contains a directed cycle)

12.2.3 Example of Testing Conflict Serializability

Schedule S1:

$$\boxed{r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B);}$$

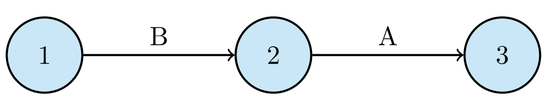

图 **12**. Precedence Graph of Schedule S1

Here, we can see that Schedule S1 is **conflict serializable** because the graph is **acyclic**.

Schedule S2:

$$\boxed{r_2(A); r_1(B); w_2(A); r_2(B); r_3(A); w_1(B); w_3(A); w_2(B)}$$

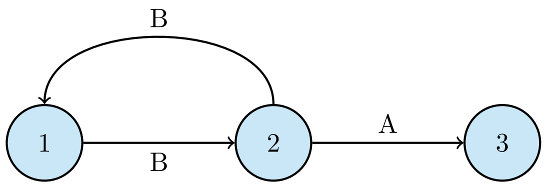

图 **13**. Precedence Graph of Schedule S2

Here, we can see that Schedule S2 is **non-serializable** because the graph is **cyclic**. There is a circle from T1 to T2 and from T2 to T1.

Section 13

Concurrency Control

We need Concurrency Control for two main reasons:

- A policy in which only one transaction can execute at a time is too restrictive(inefficient, poor degree of Concurrency).
- Testing a schedule for serializability after it has executed is too late and too expensive.

And the goal of Concurrency Control is to develop control protocols to ensure **serializability** while allowing **maximum concurrency**.

Subsection 13.1

Concurrency Control vs. Serializability Tests

- Concurrency-control protocols **do not examine the precedence graph**. Instead, a **protocal** imposes a discipline that avoids non-serializable schedules

Subsection 13.2

Locks

A lock is a mechanism to control concurrent access to a data item.
And. there are two types of locks:

1. **Exclusive(X) Lock**:
   - The data item can be both read as well as wirtten.

- Also known as a **write lock**.

2. **Shared(S) Lock**:

   - The data item can only be read.
   - Also known as a **read lock**.

Each transaction can only read/write a data item after the lock is granted.

Subsection 13.3

Lock-Based Protocols

An exclusive lock is requested using **write_lock()** function.
An shared lock is requested using **read_lock()** function.
A transaction may be granted a lock if the requested lock is **compatible** with locks already held by other transactions.

- Any number of transactions can hold shared locks on an item.
- But if a transaction holds an exclusive lock on an item, no other transaction can hold a lock on that item.

13.3.1 Local Protocol1: A simple Locking Protocol

No matter T has one or several operations manipulating X, we obtain only one lock:

- if **all** operations are read: obtain a read lock on X before reading
- if there is **at least one** write operation: obtain a write lock on X before writing
- unlock X after last operation on X

Simple locking protocol in action:

	T2	T1
t1	read_lock(Y)	
t2	read(Y)	
t3	unlock(Y)	
t4		read_lock(X)
t5		read(X)
t6		unlock(X)
t7		write_lock(Y)
t8		read(Y)
t9		$Y \leftarrow Y + X$
t10		write(Y)
t11		unlock(Y)
t12	write_lock(X)	
t13	read(X)	
t14	$X \leftarrow X + Y$	
t15	write(X)	
t16	unlock(X)	

表 **7**. Schedule S

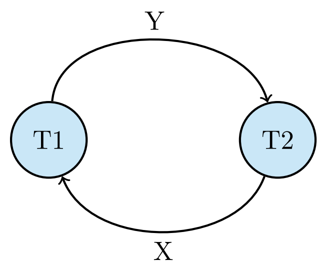

图 **14**. Precedence Graph of Schedule S

From the precedence graph, we know that Schedule S is **non-serializable**.

13.3.2 Locking Protocol2: Two-Phase Locking Protocol(2PL)

Locking protocol 2:

1. **Growing Phase**: A transaction may obtain locks, but may not release any lock.
2. **Shrinking Phase**: A transaction may release locks, but may not obtain any new lock.

Example of transactions performing 2PL:

T3	T4
write_lock(B);	read_lock(A);
read(B);	read(A);
B := B - 50;	read_lock(B);
write(B);	read(B);
write_lock(A);	display(A + B);
read(A);	unlock(A);
A := A + 50;	unlock(B);
write(A);	
unlock(B);	
unlock(A);	

表 **8**. Schedule in 2PL

Locking as above is **sufficient** to guarantee serializability.
But it may casue **deadlocks**.

SUBSECTION 13.4

Deadlocks

Definition 22 **Deadlock**
A set of transactions is deadlocked if every transaction in the set is waiting for another transaction in the set to release a lock.

13.4.1 Example of Deadlock

T3	T4
write_lock(B)	
read(B);	
B := B - 50;	
write(B);	
	read_lock(A)
	read(A)
	read_lock(B)
write_lock(A)	
...	...

表 **9**. Schedule in Deadlock

In this case:

- T3 is waiting for T4 to release the lock on A.
- T4 is waiting for T3 to release the lock on B.

Such a situation is called a **deadlock**.
Because neither T3 nor T4 can proceed.

13.4.2 Deadlock Prevention Scheme

Timeouts:
If a transaction has been waiting for a lock for a long time, it is assumed that the transaction is deadlocked, and then the system will abort it.

Pro : Small overhead and easy to implement.

Con : There may not be a deadlock. The transaction may be slow just because of the high system load.

13.4.3 Testing for Deadlocks

Wait-for Graph:

- A directed graph $G = (V, E)$
- Each transaction is represented by a node in V.
- An edge $T_i \to T_j$ exists if and only if Tj is waiting for a data item locked by Ti.

Note: The direction of the edge from wait-for graph is opposite to the direction of the edge from precedence graph.

Conclusion: If the wait-for graph has a cycle, then there is a deadlock.

Example of Wait-for Graph:

T3	T4
write_lock(B)	
read(B);	
B := B - 50;	
write(B);	
	read_lock(A)
	read(A)
	read_lock(B)
write_lock(A)	
...	...

表 **10**. Schedule in Deadlock

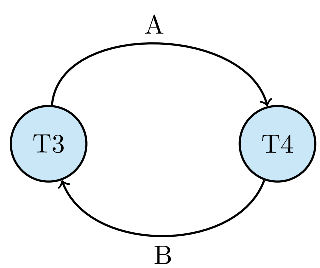

图 **15**. Wait-for Graph of Schedule S

Section 14

Database Recovery

Subsection 14.1

Transaction Failures

If a transaction fails, the database may be left in an inconsistent state. So we need to **undo** the effect of this transaction to ensure the aotmicity property.

Atomicity: Either all operations of the transaction are peoperly relfected in the database, or none are.

In order to remove inconsistency and rollback an unsuccessful transaction, we need to keep track of the changes made by the transaction.

Subsection 14.2

System Log

Definition 23 **Log records:**

1. [start transaction, T]: T starts
2. [read item, T, X]: T reads X
3. [write item, T, X, old value, new value]: T has changed the data of X from old value to new value
4. [commit, T]: T has completed successfully, and confirms all its effect can be committed to the database
5. [abort, T]: T has failed, and all its effect must be undone

Subsection 14.3

Transaction Roll Back

If a transaction fails after updating the database, but before it commits, we need to **undo** the effect of this transaction, which called transaction **roll back**.

14.3.1 Undo and Redo

Procedure UNDO(WRITE_OP):

- Undoing a write_item operation WRITE_OP
- Examing its log entry [write_item, T, X, old_value, new_value], and setting the value of X to old_value.
- Undoing must proceed in the reverse order of the operations.

Procedure REDO(WRITE_OP):

- Redoing a write_item operation WRITE_OP
- Examing its log entry [write_item, T, X, old_value, new_value], and setting the value of X to old_value.
- Redoing must proceed in the forward order of the operations.

14.3.2 Write-Ahead Logging

Write-ahead log strategy:

- Must **force** the **log record** for an update **before** correspoing data page gets to the disk.

- Must **force all records** for a transaction **before** commits.

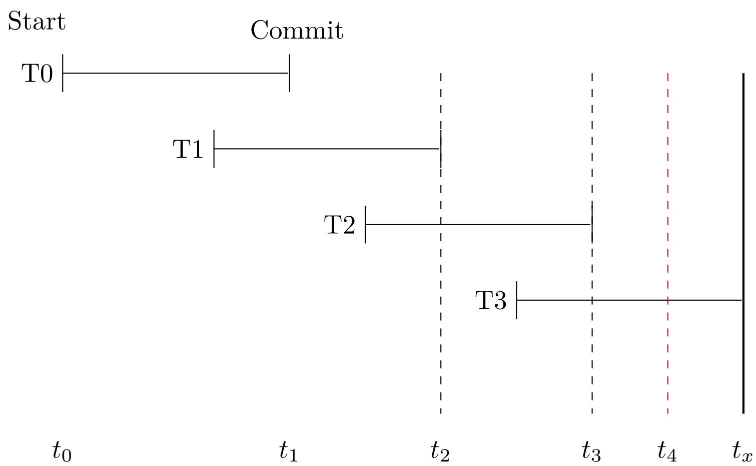

图 **16**. System Log

If there is a crash at t_4, we need to redo T_0, T_1 and T_2, and undo T_3.
Because, T_0, T_1 and T_2 have been committed, and T_3 has not been committed.

STEP1 Undo the value written by T_3 to the old data value from log.(With red line)

STEP2 Redo the value written by T_0, T_1 and T_2 to the new data value from log.(With green line)

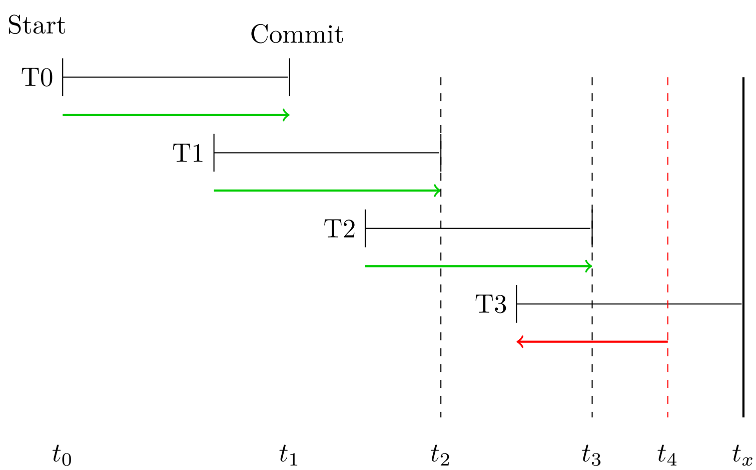

图 **17**. System Log

Subsection 14.4

Checkpoints

To reduce this problem, the system could take checkpoints at regualr intervals.
Taking a checkpoint consists of the following actions:

- Suspend execution of transactions temporarily.
- Force-write all main memory buffers to disk.
- Write a checkpoint record to the log.
- Resume execution of transactions.

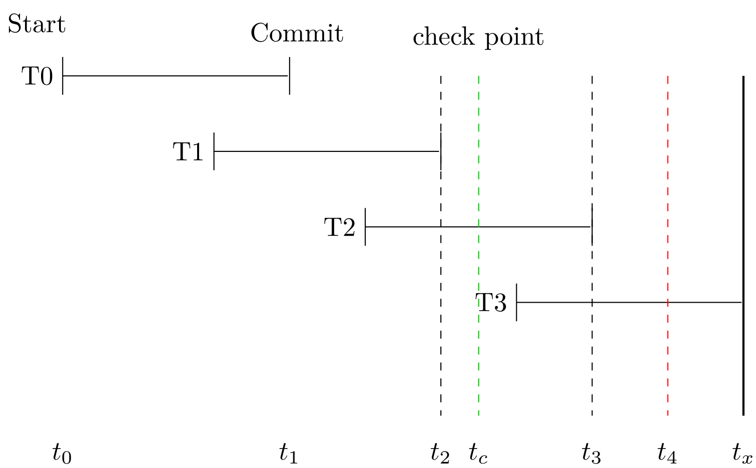

图 **18**. System Log

In the above case, we add a checkpoint at t_c.
Now if there is a crash at t_4, because at t_c, T_0 and T_1 have been committed. So we just need to undo T_3 and then redo T_2.

Week9 Graph Database

PART

XIII

好的学完了