Constraints

Inherent and implicit constraints: limitation of data model itself, e.g. list is not allowed as a valid value

Schema-based and explicit: expressed in the schema provided by the model. (E.g. cardinality in ER model)

Application based or semantic constraints: specified and enforced by the application programs

Three main types of explicit schema-based) constraints

Key constraints

1. Superkey: must be unique, an extension of key, any set of attributes including a key is a superkey e.g. {SerialNo, Make}
2. Key: minimal superkey e.g. {SerialNo} (removal of any attribute from K results in a set of attributes that is not a superkey)
3. Primary key: chosen arbitrarily from candidate keys (to uniquely identify each tuple)

Entity integrity constraints

1. primary key cannot be null ($t[PK] \neq null$ for any tuple t in r(R))

Referential integrity constraints

1. A constraint involving two relations (Used to specify a relationship among tuples in two relations: The referencing relation and the referenced relation.)
2. Referencing relation R1 have FK (foreign key) that reference the primary key PK of the referenced relation R2 ($t1[FK] = t2[PK]$)
3. Can be an existing primary key in referenced relation or null

+ Domain constraint (Every tuple value must be from the domain of its attribute)

+ Semantic Integrity Constraints (based on application semantics that cannot be expressed by the model)

Relational Database Schema: A set S of relation schemas that belong to the same database.

Relational database state is a set of relation states such that all relation states satisfy the integrity constraints (A database state that does not meet the constraints is an invalid state)

Whenever the database is changed, a new state arises

Basic operations for changing the database:

INSERT a new tuple in a relation

DELETE an existing tuple from a relation

MODIFY an attribute of an existing tuple

- Relational Database Design

Semantics must be clear:

Attributes of different entities shouldn't be mixed in the same relation, use foreign key instead

Entity and relationship attributes should be kept apart

Redundant Information:

Storing info redundantly will cause update anomalies (data inconsistency that results from data redundancy and partial update)

Null Values in Tuples :

Relations should be designed that tuples have as few NULL values as possible

Attributes that are NULL frequently could be placed in separate relations (with the primary key)

Avoid Generation of Spurious Tuples:

Natural join should be lossless: not produce Spurious(伪) tuple

two important properties of decompositions:

1. Non-additive or loss lessness of the corresponding join
2. Preservation of the functional dependencies.

- Functional Dependencies (FD):

specify formal measures of the "goodness" of relational designs

A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y

X -> Y holds if whenever two tuples have the same value for X, they must have the same value for Y (For any two tuples t1 and t2 in any relation instance r(R): if t1[X]=t2[X], then t1[Y]=t2[Y])

rl e.g. SSN -> ENAME (Social security number determines employee name)

PNUMBER -> {PNAME, PLOCATION} (Project number determines project name and location)

- **Normalization** is the process of splitting relations into well structured relations that allow users to insert, delete, and update tuples without introducing database

Normal forms - Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

1NF (First Normal Form) Disallows:

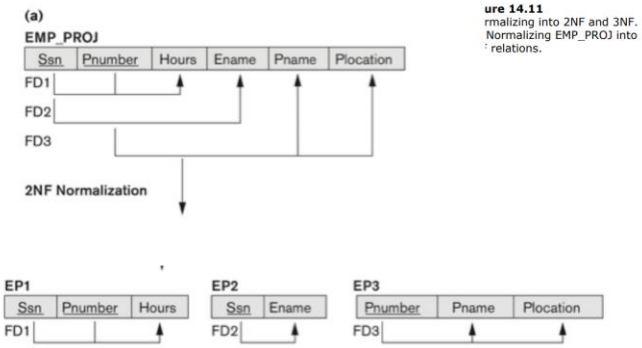
composite attributes

multivalued attributes: e.g. Location was {A,B,C}, now split and make three tuples with location A, B and C

nested relations: attributes whose values for an individual tuple are non-atomic (break one relation with nested relation into two relations sharing primary key)

- 2NF (Second Normal Form)

it is in 1NF and every non-prime attribute of the relation is dependent on the whole of every candidate key (break down Functionality dependence)



ure 14.11
rmalizing into 2NF and 3NF.
Normalizing EMP_PROJ into
relations.

Prime attribute must be a member of some candidate key

Full functional dependency: a FD Y -> Z where removal of any attribute from Y means the FD does not hold any more

e.g. PNUMBER = project number

{SSN, PNUMBER} -> HOURS is a full FD since neither SSN -> HOURS nor PNUMBER -> HOURS hold

- 3NF (Third Normal Form)

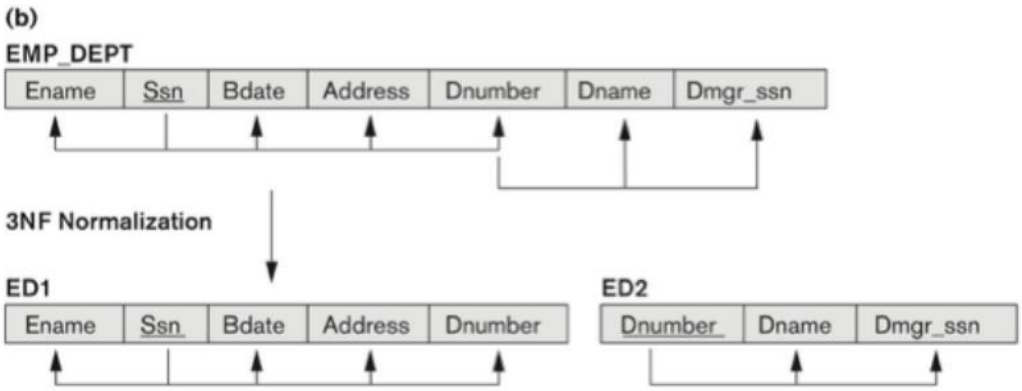
if it is in 2NF and non-prime attribute A in R is NOT transitively dependent on the primary key (whenever a FD X → A holds in R, then either: (a) X is a superkey of R, or (b) A is a prime attribute of R)

Note: In X -> Y and Y -> Z, with X as the primary key, we consider this a problem only if Y is not a candidate key.

Transitive functional dependency: a FD X -> Z that can be derived from two FDs X -> Y and Y -> Z

If SSN -> DNUMBER and DNUMBER -> DMGRSSN

Then SSN -> DMGRSSN



- 1st normal form
  - All attributes depend on **the key**
- 2nd normal form
  - All attributes depend on **the whole key**
- 3rd normal form
  - All attributes depend on **nothing but the key**

- BCNF (Boyce-Codd Normal Form) - Lossless, dependency-preserving decomposition into BCNF MAY NOT BE possible.

whenever an FD X → A holds in R, then X is a superkey of R

These rules are sound and complete

- Armstrong's inference rules:
  - IR1. (**Reflexive**) If Y *subset-of* X, then X → Y
  - IR2. (**Augmentation**) If X → Y, then XZ → YZ
    - (Notation: XZ stands for X U Z)
  - IR3. (**Transitive**) If X → Y and Y → Z, then X → Z

Some additional inference rules that are useful:

- **Decomposition:** If X → YZ, then X → Y and X → Z
- **Union:** If X → Y and X → Z, then X → YZ
- **Psuedotransitivity:** If X → Y and WY → Z, then WX → Z

Closure: the complete set of all possible attributes that can be functionally derived from functional dependency using the inference rules

Step 1; **Add attributes present on the LHS of the first functional dependency to the closure.**

Step-2 : Add attributes present on the RHS of the original functional dependency to the closure.

Step-3 : Add the other possible attributes which can be derived using attributes present on the RHS of the closure.

Example-2 : Consider another relation R(A, B, C, D, E) having the Functional dependencies :

FD1 : A → BC

FD2 : C → B

FD3 : D → E

FD4 : E → D

Now, calculating the closure of the attributes as :

$\{A\}^+ = \{A, B, C\}$

$\{B\}^+ = \{B\}$

$\{C\}^+ = \{C, B\}$

$\{D\}^+ = \{E, D\}$

$\{E\}^+ = \{E, D\}$

Cover:

F covers G if every FD in G can be inferred from F n (i.e., if G+ subset-of F+)

F and G are equivalent if F covers G and G covers F

1. **Prime Attributes : Attributes which are indispensable part of candidate keys. For example : "A, D, E" attributes are prime attributes in above example-2.**
2. **Non-Prime Attributes : Attributes other than prime attributes which does not take part in formation of candidate keys. For example.**
3. **Extraneous Attributes : Attributes which does not make any effect on removal from candidate key.**

Minimal cover

Minimal cover

e.g. F = {A->D BC->AD C->B E->A E->D}

step 1: make sure all FD is single RHS

F = {A->D BC->A BC->D C->B E->A E->D}

Step 2: make sure no Extraneous (look for LHS with multiple elements)

First eliminate BC->A:

Look for closures:

B+: B

C+: CBAD, since C has B, we can eliminate B from LHS

Now C->A

Then eliminate BC->D:

Look for closures:

B+: B

C+: CBAD, since C has B, we can eliminate B from LHS

Now C->D

Set looks like F = {A->D C->A C->D C->B E->A E->D}

Step 3: hide a FD, check if its closure will change, if not, eliminate

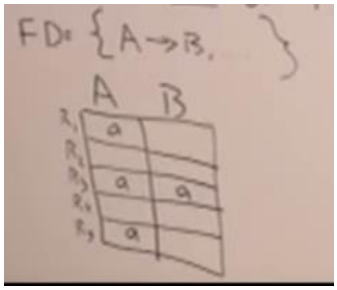
A -> D, without it A+ = A (no more D), so its needed

C -> D, without it C+ = CABD (still has D), so eliminate

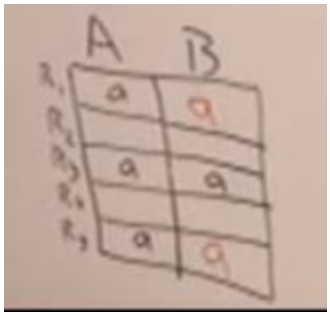
Loss-less decomposition

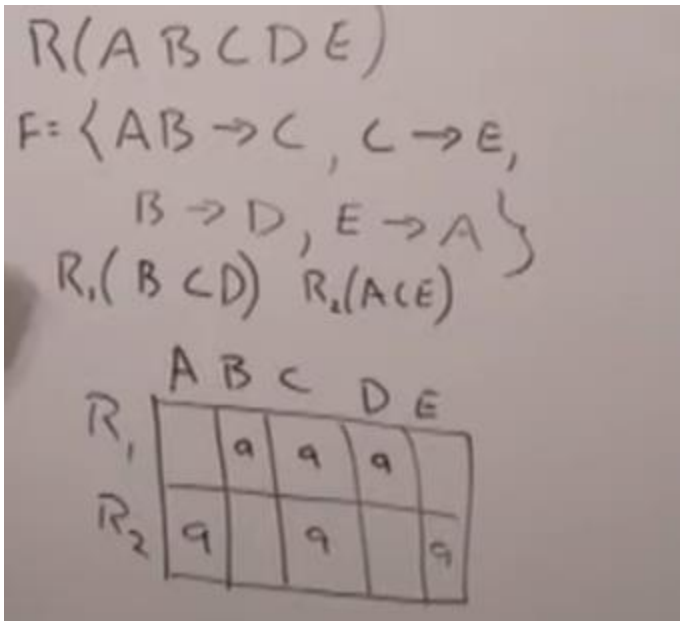
How to add distinguished var

1. 2+ rows with distinguished var for LHS
2. 1+ row with distinguished var for RHS
3. 1+ row with non-distinguished var for RHS



----- satisfy all 3 conditions ===>





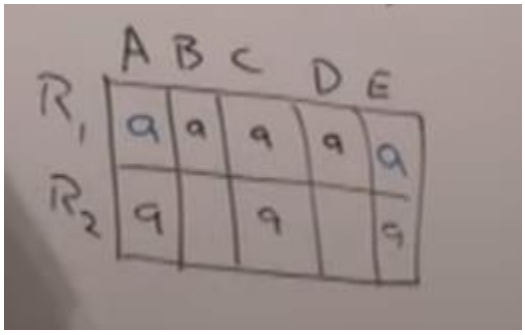
AB ->C doesn't satisfy rule 3

C->E satisfy all conditions, add dis var into RHS E

B->D doesn't satisfy rule 1

E -> A satisfy, add dis var into RHS A

Now table is like this, all distinguished var in row 1, so its lossless



Transaction

- Processes C and D in figure below

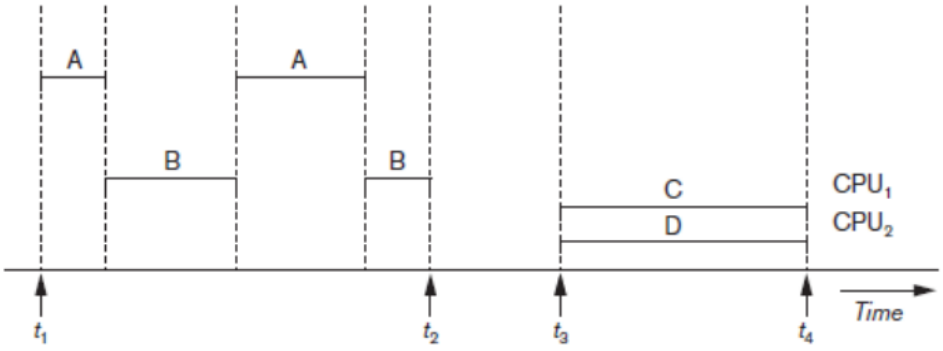


Figure 20.1 Interleaved processing versus parallel processing of concurrent transactions

Size of a data item called its granularity 粒度

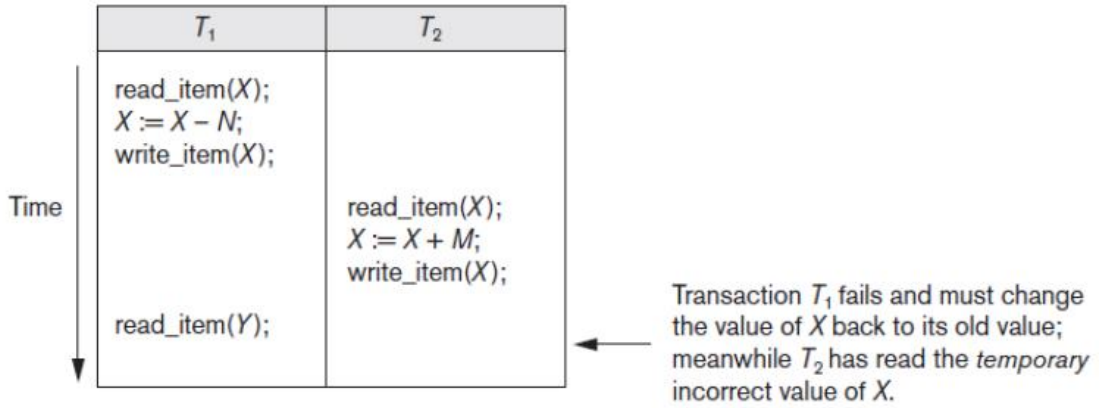
1. lost update (overwrite)： update made by one trans is overwritten by another trans

T_1	T_2
read_item(X);	
$X := X - N$;	
	read_item(X);
	$X := X + M$;
write_item(X);	
read_item(Y);	
	write_item(X);
$Y := Y + N$;	
write_item(Y);	

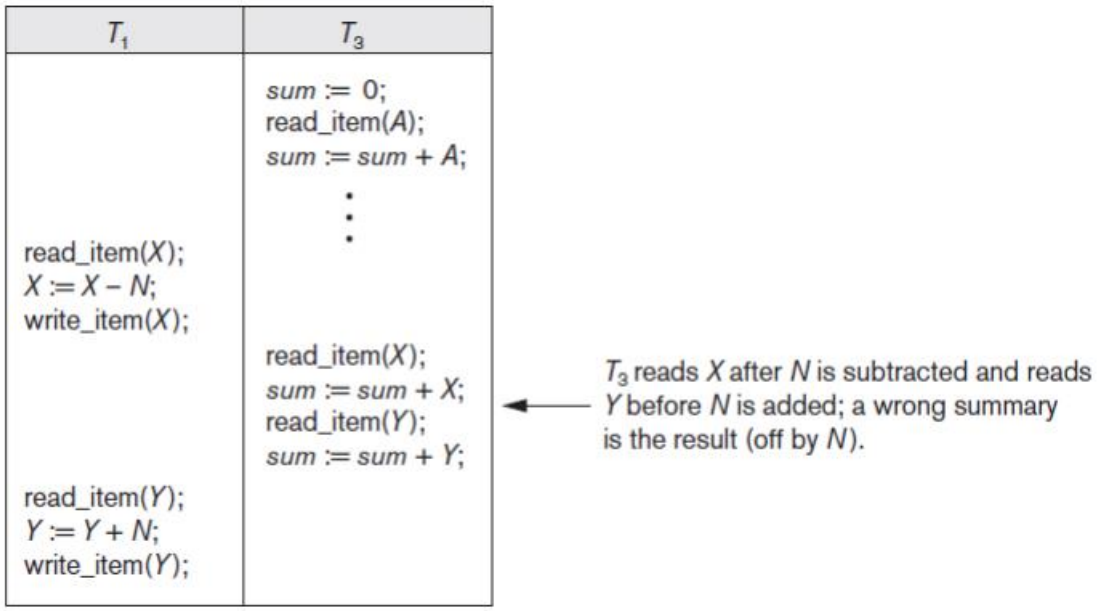
Time ↓

← Item X has an incorrect value because its update by T_1 is *lost* (overwritten).

2. Temporary Update - *dirty read*– reading uncommitted data (which might be failure and incorrect data)



Incorrect Summary



Unrepeatable Read

Transaction T reads the same item twice

Value is changed by another transaction T' between the two reads

---

Why Recovery is Needed

Committed transaction - Effect recorded permanently in the database

Aborted transaction - Does not affect the database

Types of transaction failures: Computer failure (system crash) , Transaction or system error , Local errors or exception conditions detected by the transaction

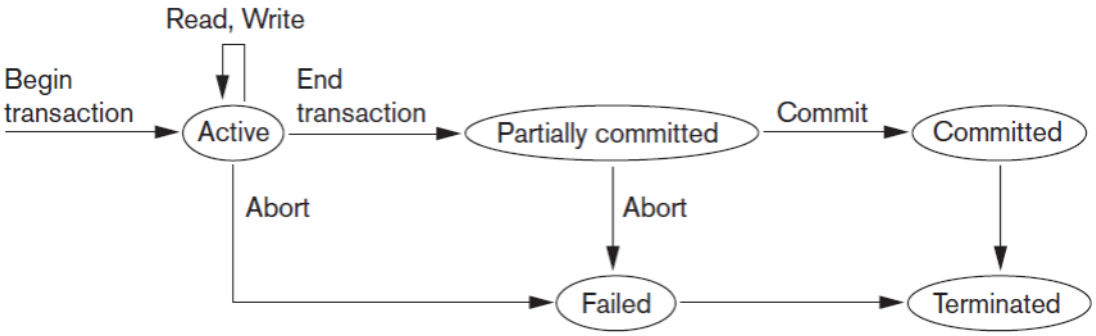
System must keep track of when each transaction starts, terminates, commits, and/or aborts
BEGIN_TRANSACTION

READ or WRITE

END_TRANSACTION

COMMIT_TRANSACTION

ROLLBACK (or ABORT)



Commit Point of a Transaction

- Occurs when all operations that access the database have completed successfully
  - And effect of operations recorded in the log
- Transaction writes a commit record into the log
  - If system failure occurs, can search for transactions with recorded start_transaction but no commit record
- Force-writing the log buffer to disk
  - Writing log buffer to disk before transaction reaches commit point

20.3 Desirable Properties of Transactions

- ACID properties
  - Atomicity
    - Transaction performed in its entirety or not at all
  - Consistency preservation
    - Takes database from one consistent state to another
  - Isolation
    - Not interfered with by other transactions
  - Durability or permanency
    - Changes must persist in the database

Conflicts: WR,RW,WW

Cascading(effect of one thing is migrate to another) rollback: when reading uncommitted data ()

A schedule S is non recoverable if T2 reads an item previously written by uncommitted T1, and commits before T1.

A schedule S is non recoverable if T2 writes an item previously written by uncommitted T1, and commits before T1.

Levels of isolation

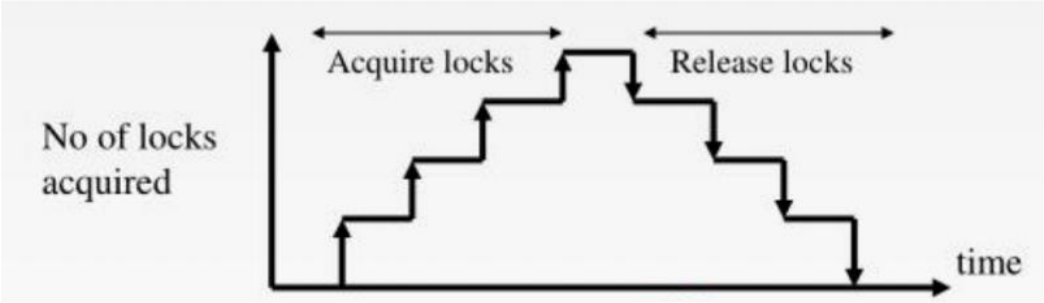
- Level 0 (read uncommitted) isolation does not overwrite the dirty reads of higher-level transactions.
- Level 1 (read committed) isolation has no dirty reads
- Level 2 (repeatable read) isolation has no lost updates and no dirty reads and repeatable read
- Level 3 (serializable) isolation has repeatable reads, phantom reads and level 2 properties

- Protocols based on a timestamp:

Older Timestamp = Higher Priority (e.g., T1 > T2)

  - Wait-Die ("Old Waits for Young")
    - If requesting T has higher priority than holding T, then requesting T waits for holding T.
    - Otherwise requesting T aborts.
  - Wound-Wait ("Young Waits for Old")
    - If requesting T has higher priority than holding T, then holding T aborts and releases lock.
    - Otherwise requesting T waits.

- Basic 2PL
  - Technique described on previous slides

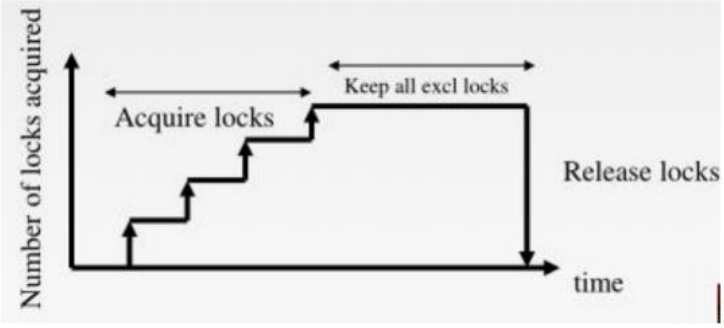


- Problems
  - Deadlocks
  - Cascading aborts.

- Conservative (static) 2PL
  - Requires a transaction to lock all the items it accesses before the transaction begins
    - Predeclare read-set and write-set
  - Deadlock-free protocol

Strict 2PL

- Transaction does not release exclusive locks until after it commits or aborts



Rigorous 2PL

- Transaction does not release any locks until after it commits or aborts

smaller timestamp means older and hence higher priority.