

# CS150A Database

Wenjie Wang

School of Information Science and Technology

ShanghaiTech University

Dec. 9, 2024

Today:

- Database Design II:
  - Functional Dependencies
  - Normalization

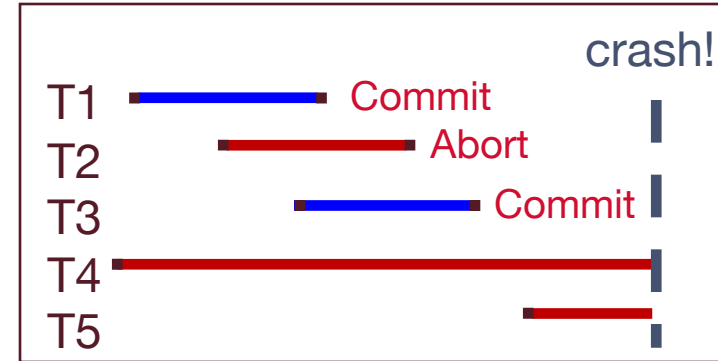
Readings:

- Database Management Systems (DBMS), Chapter 19

# Review: Recovery

# Motivation

- Atomicity:
  - Transactions may abort (“Rollback”).
- Durability:
  - What if DBMS stops running?
- Desired state after system restarts:
- T1 & T3 should be **durable**.
- T2, T4 & T5 should be **aborted** (effects not seen).
- Questions:
  - Why do transactions abort?
  - Why do DBMSs stop running?



# Buffer Management summary

	No Steal	Steal
No Force		Fastest
Force	Slowest	

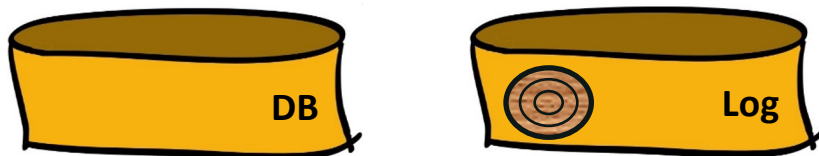
Performance  
Implications

	No Steal	Steal
No Force	No UNDO REDO	UNDO REDO
Force	No UNDO No REDO	UNDO No REDO

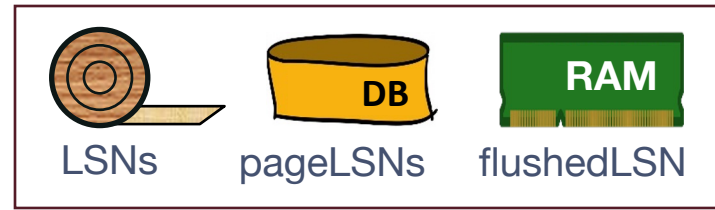
Logging/Recovery  
Implications

# Write-Ahead Logging (WAL)

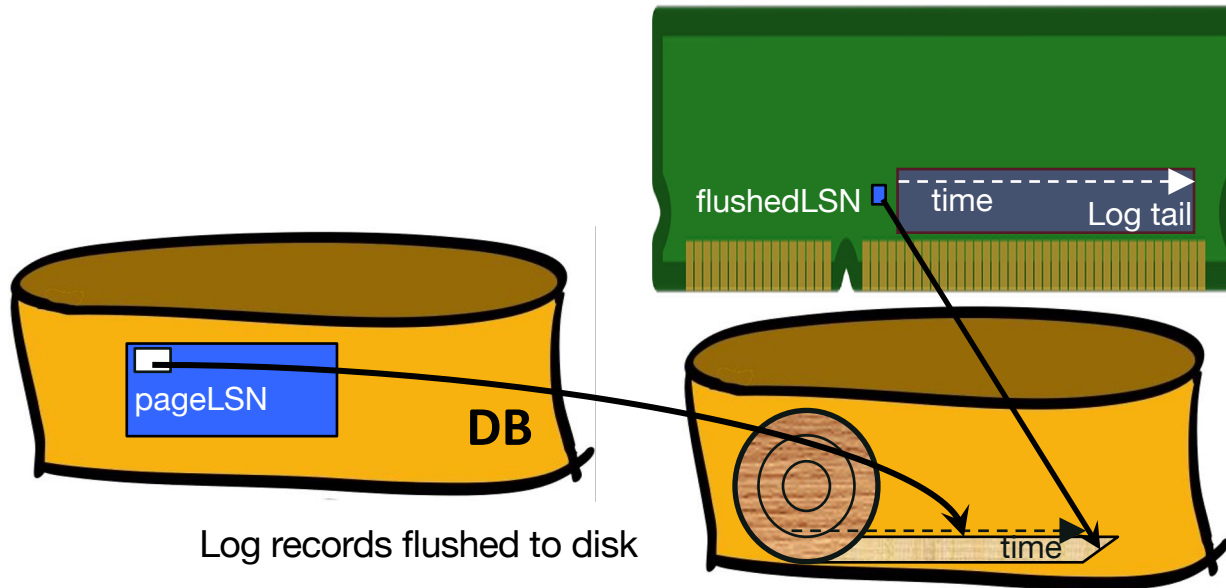
- The **Write-Ahead Logging Protocol**:
  1. Must **force** the **log record** for an update **before** the corresponding **data page** gets to the DB disk.
  2. Must **force all log records** for a Xact **before commit**.
    - I.e. transaction is not committed until all of its log records including its “commit” record are on the stable log.
- #1 (with **UNDO** info) helps guarantee Atomicity.
- #2 (with **REDO** info) helps guarantee Durability.
- This allows us to implement Steal/No-Force



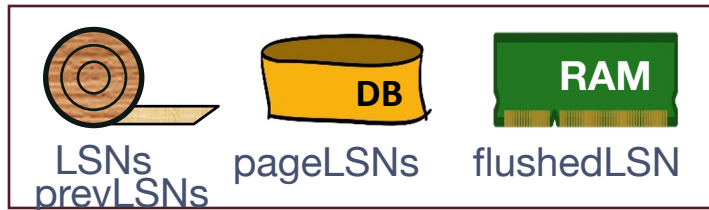
# WAL & the Log, Pt 3



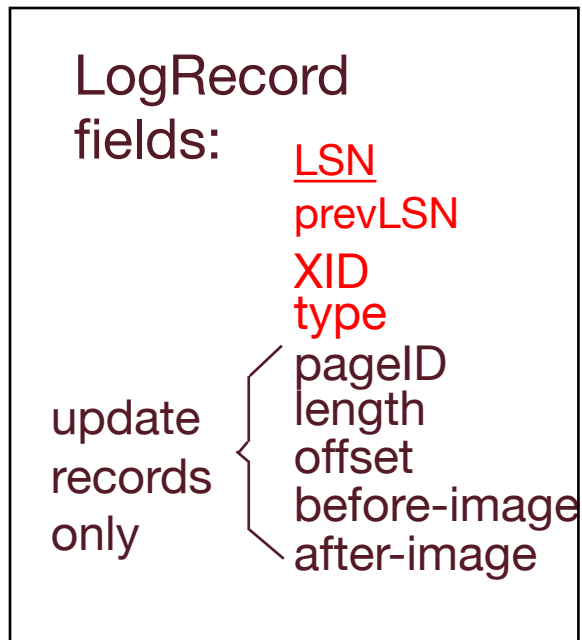
- Each **data page** in the DB contains a pageLSN.
  - A “pointer” into the log
  - The LSN of the most recent log record for an update to that page.



# Log Records, Pt 3



- Update records contain sufficient information for **REDO** and **UNDO**
  - Our “physical diff” to the left works fine.
  - There are other encodings that can be more space-efficient



# Other Log-Related State

- Two in-memory tables:
- Transaction Table
  - One entry per currently active Xact.
    - removed when Xact commits or aborts
  - Contains:
    - **XID**
    - **Status** (running, committing, aborting)
    - **lastLSN** (most recent LSN written by Xact).
- Dirty Page Table
  - One entry per dirty page currently in buffer pool.
  - Contains **recLSN**
    - LSN of the log record which first caused the page to be dirty.

Transaction Table

<u>XID</u>	Status	lastLSN
1	R	33
2	C	42

Dirty Page Table

<u>PageID</u>	recLSN
46	11
63	24



# ARIES Big Picture: What's Stored Where



LogRecords

LSN

prevLSN

XID

type

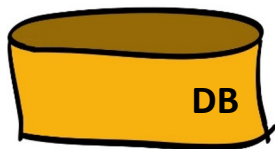
pageID

length

offset

before-image

after-image



Data pages

each with a

pageLSN

Master record



Xact Table

xid

lastLSN

status

Dirty Page Table

pid

recLSN

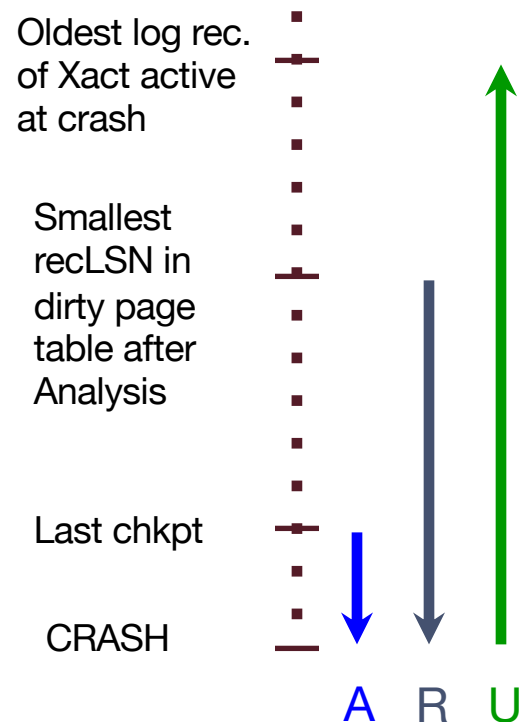
Log tail

flushedLSN

Buffer pool

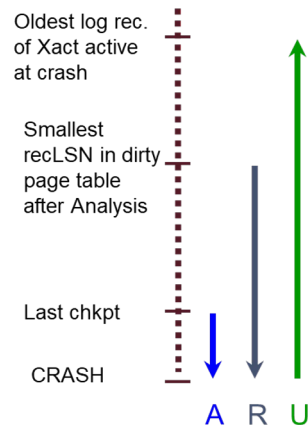
# Crash Recovery: Big Picture

- Start from a **checkpoint**
  - found via master record.
- Three phases. Need to do:
  - **Analysis** - Figure out which Xacts committed since checkpoint, which failed.
  - **REDO** all actions.
    - (repeat history)
  - **UNDO** effects of failed Xacts.



# Recovery: The Analysis Phase

- Re-establish knowledge of state at checkpoint.
  - via transaction table and dirty page table stored in the checkpoint
- Scan log forward from checkpoint.
  - **End** record:
    - Remove Xact from Xact table
  - **Update** record:
    - If page P not in Dirty Page Table, Add P to DPT, set its **recLSN=LSN**.
  - **!End** record:
    - Add Xact to Xact table
    - set lastLSN=LSN
    - change Xact status on commit.
- At end of Analysis...
  - For any Xacts in the Xact table in Committing state,:
    - Write a corresponding END log record
    - ...and Remove Xact from Xact table.
  - Now, Xact table says which xacts were active at time of crash.
    - Change status of running xacts to aborting and write abort records
  - DPT says which dirty pages might not have made it to disk



## Log

LSN	Record	prevLSN
10	T1 updates P3	null
20	T1 updates P1	10
30	T2 updates P2	null
40	T3 updates P1	null
50	Begin Checkpoint	-
60	T3 updates P3	40
70	T3 Aborts	60
80	End Checkpoint	-
90	CLR undo T3 LSN: 60	70
100	T1 updates P4	20
110	T1 commits	100
120	T1 Ends	110

## Transaction Table

Transaction	Status	lastLSN
T1	running	20
T2	running	30
T3	running	40

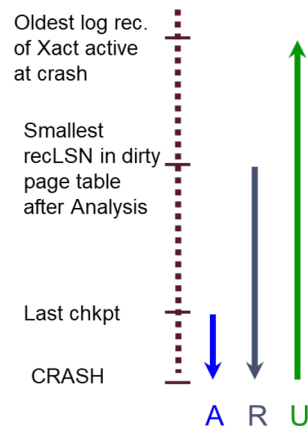
## Dirty Page Table

Page ID	recLSN
P1	40
P3	10

Transaction	Status	lastLSN	PageID	recLSN
T2	Running	30	P1	40
T3	Aborting	90	P3	10
			P4	100

# Phase 2: The REDO Phase

- We **Repeat History** to reconstruct state at crash:
  - Reapply **all** updates (even of aborted Xacts!), redo CLR's.
- Scan forward from log rec containing smallest recLSN in DPT.
  - Q: why start here?
- For each update log record or CLR with a given LSN, **REDO** the action unless:
  - Affected page is not in the Dirty Page Table, or
  - Affected page is in D.P.T., but has recLSN > LSN, or
  - pageLSN (in DB) >= LSN. (this last case requires I/O)
- To REDO an action:
  - Reapply logged action.
  - Set pageLSN to LSN. No additional logging, no forcing!



## Log

LSN	Record	prevLSN
10	T1 updates P3	null
20	T1 updates P1	10
30	T2 updates P2	null
40	T3 updates P1	null
50	Begin Checkpoint	-
60	T3 updates P3	40
70	T3 Aborts	60
80	End Checkpoint	-
90	CLR undo T3 LSN: 60	70
100	T1 updates P4	20
110	T1 commits	100
120	T1 Ends	110

Transaction	Status	lastLSN
T2	Running	30
T3	Aborting	90

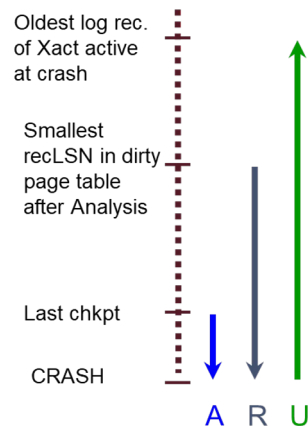
<u>PageID</u>	recLSN
P1	40
P3	10
P4	100

- 10 - UPDATE that does not meet any of the conditions
- Not 20 -  $\text{recLSN} > \text{LSN}$
- Not 30 - page not in DPT
- 40 - UPDATE that does not meet any of the conditions
- Not 50 - only redo UPDATES and CLRs
- 60 - UPDATE that does not meet any of the conditions
- Not 70 - only redo UPDATES and CLRs
- Not 80 - only redo UPDATES and CLRs
- 90 - CLR that does not meet any of the conditions
- 100 - UPDATE that does not meet any of the conditions
- Not 110 - only redo UPDATES and CLRs
- Not 120 - only redo UPDATES and CLRs

For a final answer of 10, 40, 60, 90, 100.

# Phase 3: The UNDO Phase

- A simple solution:
  - The xacts in the Xact Table are losers.
  - For each loser, perform simple transaction abort (start or continue xact rollback)
  - Problem?
    - Lots of random I/O in the log following undoNextLSN chains.
    - Can we do this in one backwards pass of log?
      - Next slide!



## Log

LSN	Record	prevLSN
10	T1 updates P3	null
20	T1 updates P1	10
30	T2 updates P2	null
40	T3 updates P1	null
50	Begin Checkpoint	-
60	T3 updates P3	40
70	T3 Aborts	60
80	End Checkpoint	-
90	CLR undo T3 LSN: 60	70
100	T1 updates P4	20
110	T1 commits	100
120	T1 Ends	110

Transaction	Status	lastLSN
T2	Running	30
T3	Aborting	90

<u>PageID</u>	recLSN
P1	40
P3	10
P4	100

<u>LSN</u>	Record	<u>prevLSN</u>	<u>undoNextLSN</u>
130	ABORT T2	30	
140	CLR Undo T3: <u>LSN</u> 40	90	null
150	END T3	140	
160	CLR Undo T2: <u>LSN</u> 30	130	null
170	END T2	160	



# Review: ER model

# Steps in Database Design

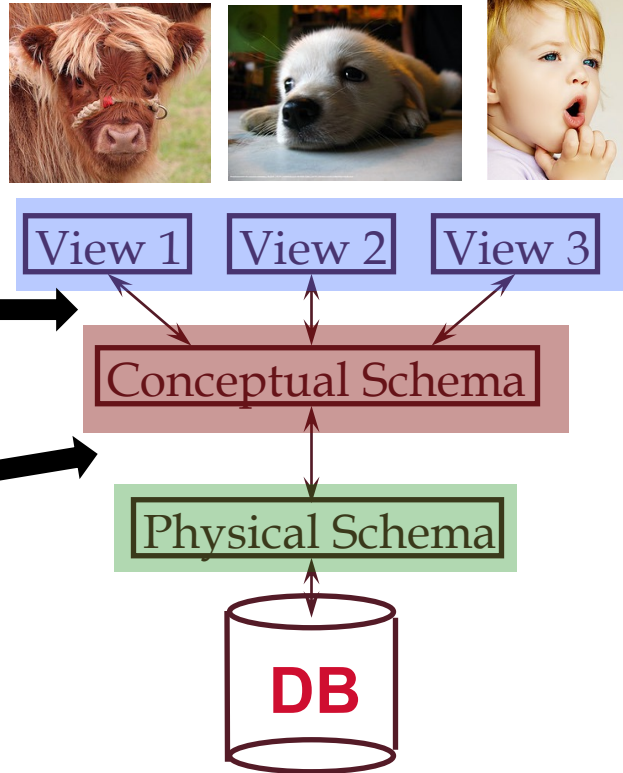
- **Requirements Analysis**
  - user needs; what must database do?
- **Conceptual Design**
  - *high level description (often done w/ER model)*
  - Object-Relational Mappings (ORMs: Hibernate, Rails, Django, etc)  
encourage you to program here
- **Logical Design**
  - translate ER into DBMS data model
  - ORMs often require you to help here too
- **Schema Refinement**
  - consistency, normalization
- **Physical Design** - indexes, disk layout
- **Security Design** - who accesses what, and how



You are here

# Review: Levels of Abstraction

**Users**



**Logical** data independence

**Physical** data independence

# Example: University Database

- **Conceptual schema:**

- Students(sid text, name text, login text, age integer, gpa float)
- Courses(cid text, cname text, credits integer)
- Enrolled(sid text, cid text, grade text)

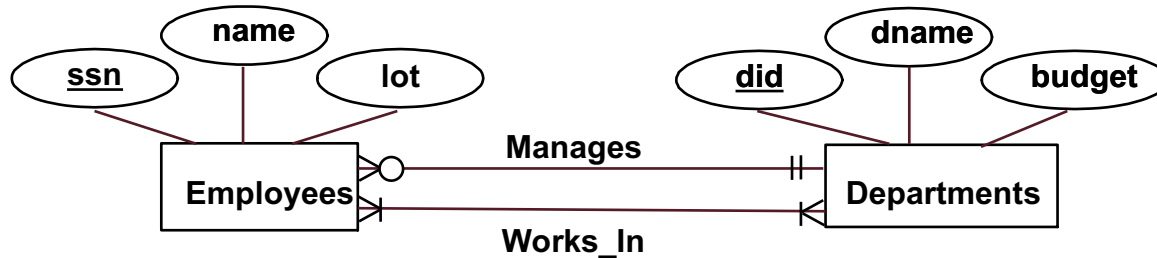
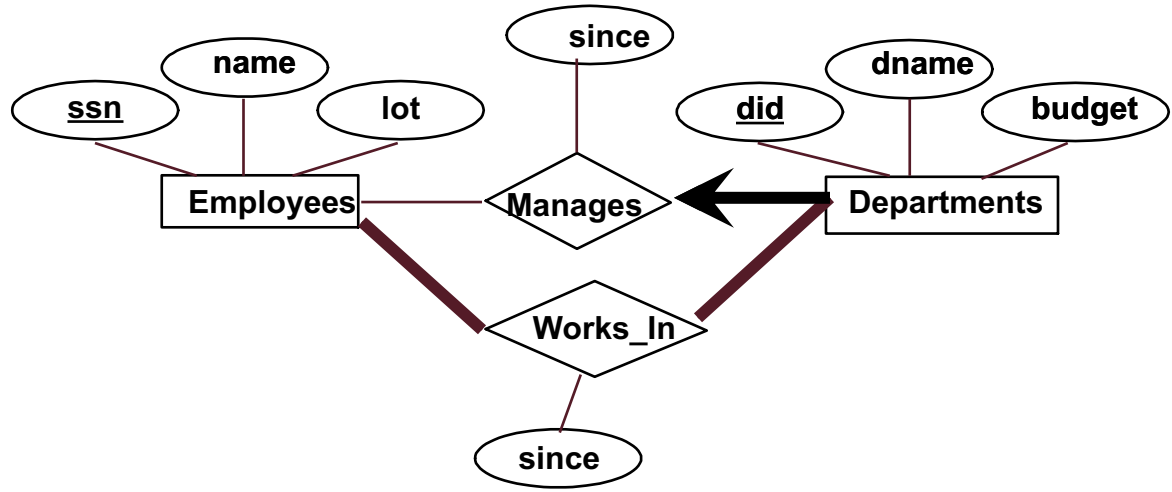
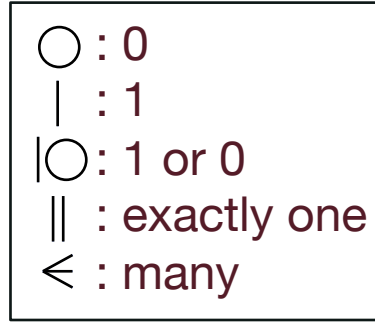
- **Physical schema:**

- Relations stored as unordered files.
- Index on first column of Students.

- **External Schema** (View):

- Course\_info(cid text, enrollment integer)

# Review: ER model



Key Constraint  
Participation Constraint  
Weak Entity

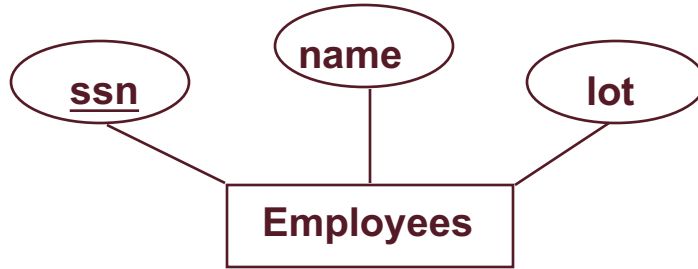
No relationship attributes

# Review: Conceptual Design Using the ER Model

- ER modeling can get tricky!
- Design choices:
  - **Entity** or **attribute**?
  - **Entity** or **relationship**?
  - Relationships: **Binary** or **ternary**? **Aggregation**?
- ER Model goals and limitations:
  - Lots of semantics can (and should) be captured.
  - Some constraints cannot be captured in ER.
    - We'll refine things in our logical (relational) design

# Review: Logical DB Design: ER to Relational

- Entity sets to tables.  
Easy.



ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

```
CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 lot INTEGER,
 PRIMARY KEY (ssn))
```



# Relationship Sets to Tables

In translating a **many-to-many** relationship set to a relation, attributes of the relation must include:

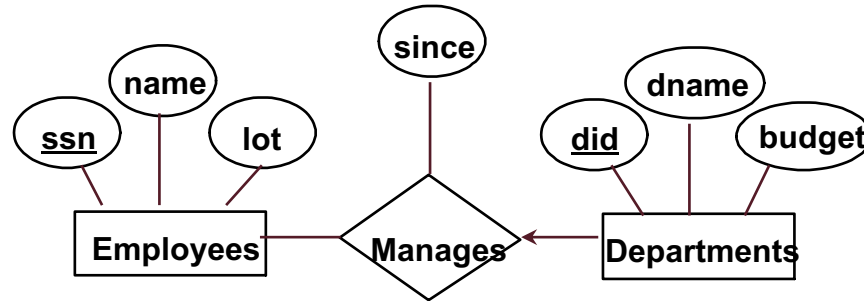
- 1) Keys for each participating entity set (as foreign keys). This set of attributes forms a **superkey** for the relation.
- 2) All descriptive attributes.

```
CREATE TABLE Works_In(  
    ssn CHAR(1),  
    did INTEGER,  
    since DATE,  
    PRIMARY KEY (ssn, did),  
    FOREIGN KEY (ssn)  
        REFERENCES Employees,  
    FOREIGN KEY (did)  
        REFERENCES Departments)
```

ssn	did	since
123-22-3666	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92



# Translating ER with Key Constraints, cont



Since each department has a unique manager, we could instead combine **Manages** and **Departments**.

```
CREATE TABLE Manages(  
  ssn CHAR(11),  
  did INTEGER,  
  since DATE,  
  PRIMARY KEY (did),  
  FOREIGN KEY (ssn)  
    REFERENCES Employees,  
  FOREIGN KEY (did)  
    REFERENCES Departments)
```

Vs.

```
CREATE TABLE Dept_Mgr(  
  did INTEGER,  
  dname CHAR(20),  
  budget REAL,  
  ssn CHAR(11),  
  since DATE,  
  PRIMARY KEY (did),  
  FOREIGN KEY (ssn)  
    REFERENCES Employees)
```

# Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints which we'll learn later).




```
CREATE TABLE Dept_Mgr(  
  did INTEGER,  
  dname CHAR(20),  
  budget REAL,  
  ssn CHAR(11) NOT NULL, -- total participation!  
  since DATE,  
  PRIMARY KEY (did),  
  FOREIGN KEY (ssn) REFERENCES Employees  
    ON DELETE NO ACTION)
```

# Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - **When the owner entity is deleted, all owned weak entities must also be deleted.**

```
CREATE TABLE Dep_Policy (  
    pname CHAR(20),  
    age INTEGER,  
    cost REAL,  
    ssn CHAR(11) NOT NULL,  
    PRIMARY KEY (pname, ssn),  
    FOREIGN KEY (ssn) REFERENCES Employees  
    ON DELETE CASCADE)
```

# Steps in Database Design, cont

- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - *high level description (often done w/ER model)*  Completed
  - ORM encourages you to program here
- Logical Design
  - translate ER into DBMS data model  Completed
  - ORMs often require you to help here too
- **Schema Refinement**  You are here
  - **consistency, normalization**
- Physical Design - indexes, disk layout
- Security Design - who accesses what, and how

# An Overview

The Evil to Avoid: Redundancy in your schema

- i.e. replicated values
- leads to wasted storage
- Far more important: insert/delete/update **anomalies**
  - Replicated data + change = Trouble.
  - We'll see examples shortly

# An Overview cont

- Solution: Functional Dependencies
  - a form of integrity constraints
  - help identify redundancy in schemas
  - help suggest refinements
- Main refinement technique: Decomposition
  - split the columns of one table into two tables
  - often good, but need to do this judiciously

# Functional Dependencies (FDs)

- Idea:  $X \rightarrow Y$  means
  - (Read “ $\rightarrow$ ” as “determines”)
  - Given any two tuples in table  $r$ ,  
if their  $X$  values are the same,  
then their  $Y$  values must be the same.  
(but not vice versa)

X	Y	Z
2	4	1
2	?	2
3	4	1

# FD's Continued

- Formally: An FD  $X \rightarrow Y$  holds over relation schema  $R$  if, for every allowable instance  $r$  of  $R$ :
  - $t1 \in r, t2 \in r, \pi_X(t1) = \pi_X(t2) \Rightarrow \pi_Y(t1) = \pi_Y(t2)$
  - ( $t1, t2$  are tuples;  $X, Y$  are sets of attributes)
- An FD is w.r.t. ***all*** allowable instances.
  - Declared based on app semantics
  - Not learned from data
    - (though you might learn suggestions for FDs)



# Important: key terminology

- Question: How are FDs related to primary keys?
  - **Primary Keys are special cases of FDs**
  - $K \rightarrow \{\text{all attributes}\}$
- Superkey: a set of columns that determines all the columns in its table
  - $K \rightarrow \{\text{all attributes}\}$ . (Also sometimes just called a key)
- Candidate Key: a **minimal** set of columns that determines all columns in its table
  - $K \rightarrow \{\text{all attributes}\}$
  - For any  $L \subset K$ ,  $L \not\rightarrow \{\text{all attributes}\}$  (minimal)
- Primary Key: a single chosen candidate key
- Index/sort “key” : columns used in an index or sort.
  - Unrelated to FDs, dependencies.

# Example: Constraints on Entity Set

- Consider relation Hourly\_Emps:  
Hourly\_Emps (ssn, name, lot, rating, wage\_per\_hr, hrs\_per\_wk)
- We can denote a relation schema by listing its attributes:
  - e.g., SNLRWH
  - This is really the set of attributes {S,N,L,R,W,H}.
- And we can use relation name to refer to the set of all its attributes
  - e.g., “Hourly\_Emps” for SNLRWH
- What are some FDs on Hourly\_Emps?
  - **ssn** is the primary key:  $S \rightarrow \text{SNLRWH}$
  - **rating** determines *wage\_per\_hr*:  $R \rightarrow W$
  - **lot** determines *lot*:  $L \rightarrow L$  (“trivial” dependency)

# So What??

- How do FDs help us think about the Evils of Redundancy?
- Let's connect FDs and Evils of Redundancy in our example...

# Problem 1 Due to $R \rightarrow W$

- **Update anomaly**: Can we modify W in only the 1st tuple of SNLRWH?

*Ha! Then that tuple will be inconsistent  
with Smiley and Madayan!*

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

# Problem 2 Due to $R \rightarrow W$

- **Insertion anomaly**: What if we want to insert an employee and don't know the hourly wage for his or her rating? (or we get it wrong?)

*Ha! Then you will have to invent a value without reference to established truth!*

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

# Problem 3 Due to $R \rightarrow W$

- **Deletion anomaly**: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

*Ha! Then you will forget established truth!*

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

# Detecting Redundancy

- Q: Why is  $R \rightarrow W$  problematic, but  $S \rightarrow W$  not?
- A: R is not a key, so any pair ((8, 10) for example) can appear multiple times.  
S is a **candidate key**, so each pair (e.g., (123-22-3666, 10)) is stored exactly once.

Hourly\_Emps

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

# Decomposing a Relation

- Redundancy can be removed by “chopping” the relation into pieces.
- FD's are used to drive this process.
  - $R \rightarrow W$  is causing the problems, so decompose SNLRWH into what relations?

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly\_Emps2

R	W
8	10
5	7

Wages



# Reasoning About FDs: Examples

- Given some FDs, we can usually infer additional FDs:
  - $\text{bookID} \rightarrow (\text{publisher}, \text{author})$  implies  $\text{bookID} \rightarrow \text{publisher}$   
and  $\text{bookID} \rightarrow \text{author}$
  - $\text{bookID} \rightarrow \text{publisher}$  and  $\text{bookID} \rightarrow \text{author}$  implies  $\text{bookID} \rightarrow (\text{publisher}, \text{author})$
  - $\text{bookID} \rightarrow \text{author}$  and  $\text{author} \rightarrow \text{publisher}$  implies  $\text{bookID} \rightarrow \text{publisher}$
- But,  
 $(\text{title}, \text{author}) \rightarrow \text{publisher}$  does NOT necessarily imply that  
 $\text{title} \rightarrow \text{publisher}$  NOR that  $\text{author} \rightarrow \text{publisher}$

# Reasoning About FDs: General

- Generally, an FD  $g$  is implied by a set of FDs  $F$  if  $g$  holds whenever all FDs in  $F$  hold.
- $F^+$  = closure of  $F$ :
  - the set of all FDs that are implied by  $F$ .
  - includes “trivial dependencies”

# Rules of Inference

- **Armstrong's Axioms** ( $X, Y, Z$  are sets of attributes):
  - Reflexivity: If  $X \supseteq Y$ , then  $X \rightarrow Y$
  - Augmentation: If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$  for any  $Z$
  - Transitivity: If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$
- Sound and complete inference rules for FDs!
  - using AA you get only the FDs in  $F^+$  and all these FDs.
- Some additional rules (that follow from AA):
  - Union: If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$
  - Decomposition: If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$
  - See if you can prove these!

# Example

- **Contracts(cid,sid,jid,did,pid,qty,value), and:**
  - C is the key:  $C \rightarrow CSJDPQV$
  - Proj (J) purchases each part (P) using single contract (C):  $JP \rightarrow C$
  - Dept (D) purchases at most 1 part (P) from a supplier (S):  $SD \rightarrow P$
- **Problem: Prove that SDJ is a key for Contracts**
- $JP \rightarrow C, C \rightarrow CSJDPQV$ 
  - Imply  $JP \rightarrow CSJDPQV$
  - (by transitivity) (shows that JP is a key)
- $SD \rightarrow P$ 
  - implies  $SDJ \rightarrow JP$  (by augmentation)
- $SDJ \rightarrow JP, JP \rightarrow CSJDPQV$ 
  - imply  $SDJ \rightarrow CSJDPQV$
  - (by transitivity) (shows that SDJ is a key).
- Q: can you now infer that  $SD \rightarrow CSDPQV$

# Attribute Closure

- Computing closure  $F^+$  of a set of FDs  $F$  is hard:
  - exponential in # attrs!
- Typically, just check if  $X \rightarrow Y$  is in  $F^+$ . Efficient!
  - Compute attribute closure of  $X$  (denoted  $X_+$ ) wrt  $F$ .  
 $X_+ =$  Set of all attributes  $A$  such that  $X \rightarrow A$  is in  $F^+$ 
    - $X_+ := X$
    - Repeat until no change (fixpoint):  
for  $U \rightarrow V \subseteq F$ ,  
if  $U \subseteq X_+$ , then add  $V$  to  $X_+$
  - Check if  $Y$  is in  $X_+$
  - Approach can also be used to check for keys of a relation.
    - If  $X_+ = R$ , then  $X$  is a superkey for  $R$ .
    - Q: How to check if  $X$  is a “candidate key” (minimal)?
    - A: For each attribute  $A$  in  $X$ , check if  $(X-A)_+ = R$

# Attribute Closure (example)

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

$F = \{ B \rightarrow CD, D \rightarrow E, B \rightarrow A, E \rightarrow C, AD \rightarrow B \}$

- **Is  $B \rightarrow E$  in  $F^+$  ?**

$B^+ = \{B, C, D, E, \dots\}$

... Yep!

- **Is  $D$  a key for  $R$ ?**

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

... Nope!

- **Is  $AD$  a key for  $R$ ?**

$AD^+ = \{A, D, E, C, B\}$

...Yep!

- **Is  $AD$  a *candidate* key for  $R$ ?**

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

...Yes!

- **Is  $ADE$  a candidate key for  $R$ ?**

No!

# Thanks for that...

- So we know a lot about FDs
- So what?
- Can they help with removing redundancy?

# The Notion of Normal Forms

- Q1: is any refinement is needed??!
- If relation is in a *normal form* (e.g. **BCNF**):
  - we know certain problems are avoided/minimized.
  - helps decide whether decomposing relation is useful.
- Consider a relation R with 3 attributes, ABC.
  - *No (non-trivial) FDs hold*: No redundancy here.
  - *Given  $A \rightarrow B$* : **If A is not a key**, then several tuples could have the same A value, and if so, they'll all have the same B value!



# Basic Normal Forms

- 1st Normal Form – all attributes atomic
  - I.e. relational model
  - Violated by many common data models
    - Including XML, JSON, various OO models
  - Some of these “non-first-normal form” (NFNF) quite useful in various settings
    - especially in update-never settings like data transmission
    - if you never “unnest”, then who cares!
      - basically relational collection of structured objects
- 1st  $\supset$  2nd (of historical interest)
  - $\supset$  3rd (of historical interest)
  - $\supset$  Boyce-Codd ...

# Boyce-Codd Normal Form (BCNF)

- Reln R with FDs F is in BCNF if, for all  $X \rightarrow A$  in  $F^+$ 
  - $A \subseteq X$  (called a trivial FD), or
  - X is a superkey for R.
- In other words: “R is in BCNF if the only non-trivial FDs over R are key constraints.”

# Why is BCNF Useful?

- If R in BCNF, every field of every tuple records useful info that cannot be inferred via FDs.
  - Say we know FD  $X \rightarrow A$  holds for this example relation:
  - Can you guess the value of the missing attribute
  - Yes, so relation is not in BCNF

X	Y	A
x	y1	a
x	y2	?

# Decomposition of a Relation Scheme

- How to normalize a relation?
  - **decompose** into multiple normalized relations
- Suppose  $R$  contains attributes  $A_1 \dots A_n$ .  
A **decomposition** of  $R$  consists of replacing  $R$  by two or more relations such that:
  - Each new relation scheme contains a **subset** of the attributes of  $R$ , and
  - Every attribute of  $R$  appears as an attribute of at least one of the new relations.

# Example (same as before)

Hourly\_Emps

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- SNLRWH has FDs  $S \rightarrow \text{SNLRWH}$  and  $R \rightarrow W$
- Q: Is this relation in BCNF?
  - No, The second FD causes a violation;
  - W values repeatedly associated with R values.

# Decomposing a Relation, Part 2

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly\_Emps2

R	W
8	10
5	7

Wages

Q: Are both of these relations are now in BCNF?

Yes!  $S \rightarrow SNLRH$  is OK, as is  $R \rightarrow W$ .

# Problems with Decompositions

- There are three potential problems to consider:
  - 1) May be ***impossible*** to reconstruct the original relation! (Lossiness)
    - Fortunately, not in the SNLRWH example.
  - 2) Dependency checking may require joins.
    - Fortunately, not in the SNLRWH example.
  - 3) Some queries become more expensive.
    - e.g., How much does Guldu earn?
- **Tradeoff**: Must consider these 3 vs. redundancy.

# Lossless Decomposition (example)

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40



R	W
8	10
5	7

Wages

Hourly\_Emps2

=

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40



# Lossy Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8



A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

$A \rightarrow B$ ;  $B \rightarrow C$

A	B
1	2
4	5
7	2



B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

# Lossless Join Decompositions

- Defn: Decomposition of R into X and Y is **lossless-join** w.r.t. a set of FDs F if, for every instance  $r$  that satisfies F:

$$\pi_X(r) \bowtie \pi_Y(r) = r$$

- It is always true that  $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$ 
  - When the relation is equality, the decomposition is lossless-join.
- Definition easily extended to decomposition into 3 or more relations
- It is essential that all decompositions used to deal with redundancy be lossless!***
- (Avoids Problem #1)

# More on Lossless Decomposition

- Theorem: The decomposition of R into X and Y is **lossless with respect to F** if *and only if* the closure of F contains:

$$X \cap Y \rightarrow X, \text{ or}$$

$$X \cap Y \rightarrow Y$$

- From example: decomposing ABC into AB and BC is lossy, because their intersection (i.e., B) is not a key of either resulting relation.
- Useful corollary: If  $X \rightarrow Z$  holds over R and  $X \cap Z$  is empty then decomposition of R into R-Z and XZ is loss-less (b/c X is a superkey of XZ!)
- Just like our BCNF example!  
Where X is Rating, Z is Wage.  
Clearly Rating intersect Wage is empty.  
So decompose into SNLRH and RW and it is lossless.

# Lossless Decomposition? Yes, but...

A	B	C
1	2	3
4	5	6
7	2	8



A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B$ ;  $C \rightarrow B$

A	C
1	3
4	6
7	8



B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8

- But, now we can't check  $A \rightarrow B$  without doing a join!

# Dependency Preserving Decomposition

- **Dependency preserving decomposition** (Intuitive):
  - A decomposition where the following is true:
    - If  $R$  is decomposed into  $X$ ,  $Y$  and  $Z$ ,
    - and we enforce FDs individually on each of  $X$ ,  $Y$  and  $Z$ ,
    - then all FDs that held on  $R$  must also hold on result.
    - (Avoids Problem #2 on our list.)
- **Defn: Projection of set of FDs  $F$  :**
  - If  $R$  is decomposed into  $X$  and  $Y$ , the projection of  $F$  on  $X$  (denoted  $F_X$ ) is the set of FDs  $U \rightarrow V$  in  $F^+$ , such that all of the attributes  $U, V$  are in  $X$ .
- *$F^+$ : closure of  $F$ , not just  $F$ !*

## Dependency Preserving Decompositions: Definition

- Defn: Decomposition of R into X and Y is **dependency preserving** if  $(F_X \cup F_Y)^+ = F^+$ 
  - i.e., if we consider only dependencies in the closure  $F^+$  that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in  $F^+$ .
  - (just the formalism of our intuition above)

# Dependency Preservation: Notes

- Critical to consider  $F^+$  in the definition:
  - ABC,  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow A$ , decomposed into AB and BC.
  - Is this dependency preserving? Is  $C \rightarrow A$  preserved????
- Well...  $F^+$  contains  $F \cup \{A \rightarrow C, B \rightarrow A, C \rightarrow B\}$ , so...
  - $F_{AB} \supseteq \{A \rightarrow B, B \rightarrow A\}$ ;  $F_{BC} \supseteq \{B \rightarrow C, C \rightarrow B\}$
  - So,  $(F_{AB} \cup F_{BC})^+ \supseteq \{B \rightarrow A, C \rightarrow B\}$
  - Hence  $(F_{AB} \cup F_{BC})^+ \supseteq \{C \rightarrow A\}$

$$(F_X \cup F_Y)^+ = F^+$$

# Decomposition into BCNF

- Consider relation R with FDs F.
- If  $X \rightarrow Y$  violates BCNF, decompose R into R - Y and XY (guaranteed to be loss-less).
  - Repeated application of this idea will give us a collection of relations that are in BCNF
  - Lossless join decomposition, and guaranteed to terminate.
- e.g., CSJDPQV, key C,  $JP \rightarrow C$ ,  $SD \rightarrow P$ ,  $J \rightarrow S$ 
  - {contractid, supplierid, projectid,deptid,partid, qty, value}
  - To deal with  $SD \rightarrow P$ , decompose into SDP, CSJDQV.
  - To deal with  $J \rightarrow S$ , decompose CSJDQV into JS and CJDQV
  - So we end up with: SDP, JS, and CJDQV
- Note: several dependencies may cause violation of BCNF.
- The order in which we “deal with” them could lead to very different sets of relations!



# BCNF and Dependency Preservation

- In general, **there may not be a dependency preserving decomposition into BCNF.**
- E.g., CSZ,  $CS \rightarrow Z$ ,  $Z \rightarrow C$ 
  - Can't decompose while preserving 1st FD; not in BCNF.
- Similarly, decomposition of CSJDPQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs  **$JP \rightarrow C$** ,  $SD \rightarrow P$  and  $J \rightarrow S$ ).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - but JPC tuples are stored only for checking the f.d. (**Redundancy!**)

# Summary of Schema Refinement

- BCNF: each field contains data that cannot be inferred via FDs.
  - ensuring BCNF is a good heuristic.
- Not in BCNF? Try decomposing into BCNF relations.
  - Must consider whether all FDs are preserved!

# Summary of Schema Refinement, cont

- What to do when a lossless-join, dependency preserving decomposition into BCNF is impossible?
  - There is a more *permissive* Third Normal Form (3NF)
    - In the hidden slides of slide deck, or book, or Wikipedia
  - But you'll have redundancy. Beware. You will need to keep it from being a problem in your application code.
- Note: even more *restrictive* Normal Forms exist
  - we don't cover them in this course, but some are in the book