Database Management Systems

Ehsan Noei
e.noei@utoronto.ca

UNIVERSITY OF TORONTO

eID	Name	Department	Location	Budget
1	Steve	Information	CA	$10
2	Nash	Information	CA	$10
3	Ehsan	Information	CA	$10
4	Amy	Information	CA	$10
5	James	CS	CA	$9

eID	Name	Department	Location	Budget
1	Steve	Information	CA	$10
2	Nash	Information	CA	$10
3	Ehsan	Information	CA	$10
4	Amy	Information	CA	$10
5	James	CS	CA	$9

Insertion Anomaly

eID	Name	Department	Location	Budget
1	Steve	Information	CA	$10
2	Nash	Information	CA	$10
3	Ehsan	Information	CA	$10
4	Amy	Information	CA	$10
5	James	CS	CA	$9
6	Joe	Information	CA	$10

eID	Name	Department	Location	Budget
1	Steve	Information	CA	$10

eID	Name	Department	Location	Budget

Deletion Anomaly

eID	Name	Department	Location	Budget

Update Anomaly

eID	Name	Department	Location	Budget
1	Steve	Information	CA	$10
2	Nash	Information	CA	$12
3	Ehsan	Information	CA	$10
4	Amy	Information	CA	$10
5	James	CS	CA	$9

Solution?

eID	Name	Department
1	Steve	Information
2	Nash	Information
3	Ehsan	Information
4	Amy	Information
5	James	CS

Department	Location	Budget
Information	CA	$10
CS	CA	$9

Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  - redundant storage
  - insert/delete/update anomalies
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD).

Functional Dependencies (FDs)

- A <u>functional dependency</u> $X \rightarrow Y$ holds over relation R if, for every allowable instance *r* of R:

  - $t1 \in r,\ t2 \in r$
  - $\pi_X(t1) = \pi_X(t2)$

  $\Big\}$ implies $\pi_Y(t1) = \pi_Y(t2)$

- i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)

Functional Dependencies (FDs)

- An FD is a statement about *all* allowable relations.
  - Must be identified based on semantics of application.
  - Given some allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!
- K is a candidate key for R means that K → R
  - However, K → R does not require K to be *minimal.*

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (*ssn*, *name*, *lot*, *rating*, *hrly_wages*, *hrs_worked*)
- *Notation*: We will denote this relation schema by listing the attributes: SNLRWH
  - This is really the *set* of attributes {S,N,L,R,W,H}.
  - Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly_Emps for SNLRWH)
- Some FDs on Hourly_Emps:
  - *ssn* is the key: $S \longrightarrow SNLRWH$
  - *rating* determines *hrly_wages*: $R \longrightarrow W$

- Problems due to R → W :
- *Update anomaly*: Can we change W in just the 1st tuple of SNLRWH?
- *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his rating?
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Will 2 smaller tables be better?

Wages

R	W
8	10
5	7

Hourly_Emps2

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

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

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - *ssn* → *did*, *did* → *lot* implies *ssn* → *lot*
- F^+ = *closure of F* is the set of all FDs that are implied by *F*.
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If X ⊆ Y, then Y → X
  - *Augmentation*: If X → Y, then XZ → YZ for any Z
  - *Transitivity*: If X → Y and Y → Z, then X → Z

Reasoning About FDs (Contd.)

- Couple of additional rules (that follow from AA):
  - *Union*: If X → Y and X → Z, then X → YZ
  - *Decomposition*: If X → YZ, then X → Y and X → Z
- Example:
  - Contracts(*contractid, supplierid, projectid, deptid, partid, qty, value)*
  - C is the key: C → CSJDPQV
  - Project purchases each part using single contract: JP → C
  - Dept purchases at most one part from a supplier: SD → P
- JP→ C, C→ CSJDPQV imply JP→ CSJDPQV
- SD→ P implies SDJ→ JP
- SDJ→ JP, JP→ CSJDPQV imply SDJ→ CSJDPQV

Reasoning About FDs (Contd.)

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs!)
- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs *F*. An efficient check:
  - Compute ***attribute closure*** of X (denoted X^+) with respect to *F:*
    - Set of all attributes A such that $X \rightarrow A$ is in F^+
    - There is a linear time algorithm to compute this.
  - Check if Y is in X^+

Reasoning About FDs (Contd.)

closure = X;

repeat until there is no change: {

if there is an FD U → V in F such that U ⊆ closure,

then set closure = closure U V·

}

Reasoning About FDs (Contd.)

- Does F = {A $\rightarrow$ C, B $\rightarrow$ C, C $\rightarrow$ D, C D $\rightarrow$ E, E $\rightarrow$ G } imply A $\rightarrow$ G?
  - i.e, is A $\rightarrow$ G in the closure F^+? Equivalently, is G in A^+?

Name	Item	Address	Supplier	Supplier Phone	Price
Steve	Bag	37 West	Amazon	+1 666	$100
Nash	Luggage	73 East	eBay	+1 667	$120
John	Bag, Sunglasses	66 North	Amazon	+1 666	$220
Emily	Sunglasses	66 South	Amazon, eBay	+1 666, +1 667	$120

Multi-Valued Attributes

Name	Item	Address	Supplier	Supplier Phone	Price
Steve	Bag	37 West	Amazon	+1 666	$100
Nash	Luggage	73 East	eBay	+1 667	$120
John	Bag, Sunglasses	66 North	Amazon	+1 666	$220
Emily	Sunglasses	66 South	Amazon, eBay	+1 666, +1 667	$120

Name	Item	Address	Supplier	Supplier Phone	Price
Steve	Bag	37 West	Amazon	+1 666	$100
Nash	Luggage	73 East	eBay	+1 667	$120
John	Bag	66 North	Amazon	+1 666	$100
John	Sunglasses	66 North	Amazon	+1 666	$120
Emily	Sunglasses	66 South	Amazon	+1 666	$120
Emily	Sunglasses	66 South	eBay	+1 667	$120

Name	Item	Address	Supplier	Supplier Phone	Price
Steve	Bag	37 West	Amazon	+1 666	$100
Nash	Luggage	73 East	eBay	+1 667	$120
John	Bag	66 North	Amazon	+1 666	$100
John	Sunglasses	66 North	Amazon	+1 666	$120
Emily	Sunglasses	66 South	Amazon	+1 666	$120
Emily	Sunglasses	66 South	eBay	+1 667	$120

Attributes depend on key

Name	Item	Address	Supplier	Supplier Phone	Price
Steve	Bag	37 West	Amazon	+1 666	$100
Nash	Luggage	73 East	eBay	+1 667	$120
John	Bag	66 North	Amazon	+1 666	$100
John	Sunglasses	66 North	Amazon	+1 666	$120
Emily	Sunglasses	66 South	Amazon	+1 666	$120
Emily	Sunglasses	66 South	eBay	+1 667	$120

Attributes depend on key

Name	Item	Address	Supplier	Supplier Phone	Price
Steve	Bag	37 West	Amazon	+1 666	$100
Nash	Luggage	73 East	eBay	+1 667	$120
John	Bag	66 North	Amazon	+1 666	$100
John	Sunglasses	66 North	Amazon	+1 666	$120
Emily	Sunglasses	66 South	Amazon	+1 666	$120
Emily	Sunglasses	66 South	eBay	+1 667	$120

Name	Item	Address	Supplier	Price
Steve	Bag	37 West	Amazon	$100
Nash	Luggage	73 East	eBay	$120
John	Bag	66 North	Amazon	$100
John	Sunglasses	66 North	Amazon	$120
Emily	Sunglasses	66 South	Amazon	$120
Emily	Sunglasses	66 South	eBay	$12

Supplier	Supplier Phone
Amazon	+1 666
eBay	+1 667

Name	Item	Address	Supplier
Steve	Bag	37 West	Amazon
Nash	Luggage	73 East	eBay
John	Bag	66 North	Amazon
John	Sunglasses	66 North	Amazon
Emily	Sunglasses	66 South	Amazon
Emily	Sunglasses	66 South	eBay

Item	Price
Bag	$100
Luggage	$120
Sunglasses	$120

Supplier	Supplier Phone
Amazon	+1 666
eBay	+1 667

Name	Item	Address	Supplier
Steve	Bag	37 West	Amazon
Nash	Luggage	73 East	eBay
John	Bag	66 North	Amazon
John	Sunglasses	66 North	Amazon
Emily	Sunglasses	66 South	Amazon
Emily	Sunglasses	66 South	eBay

Item	Price
Bag	$100
Luggage	$120
Sunglasses	$120

Supplier	Supplier Phone
Amazon	+1 666
eBay	+1 667

Name	Item	Supplier
Steve	Bag	Amazon
Nash	Luggage	eBay
John	Bag	Amazon
John	Sunglasses	Amazon
Emily	Sunglasses	Amazon
Emily	Sunglasses	eBay

Name	Address
Steve	37 West
Nash	73 East
John	66 North
Emily	66 South

Supplier	Supplier Phone
Amazon	+1 666
eBay	+1 667

Item	Price
Bag	$100
Luggage	$120
Sunglasses	$120

Normal Forms

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: There is no redundancy here.
    - Given A → B: Several tuples could have the same A value, and if so, they'll all have the same B value!

Boyce-Codd Normal Form (BCNF)

- Relationship R with FDs *F* is in BCNF if, for all $X \rightarrow A$ in F^+
  - $A \in X$ (called a *trivial* FD), or
  - X contains a key for R (super key).
- In other words, R is in BCNF if the only (non-trivial) FDs that hold over R are key constraints.
  - If we are shown two tuples that agree upon the X value, we can infer the A value in one tuple from the A value in the other.

Third Normal Form (3NF)

- Relationship R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in F^+
  - $A \in X$ (called a *trivial* FD), or
  - X contains a key for R, or
  - A is part of some key for R.
- *Minimality* of a key is crucial in third condition above!
- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomp, or performance considerations).

- R: SBDC
- FD S → C
- S is not the key
- C is not part of a key
- (S, C) redundant

- R: SBDC
- FD S → C
- S is not the key
- C is part of a key

Decomposition of a Relation Scheme

- Suppose that relation R contains attributes *A1 ... An.* 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 no attributes that do not appear in R), and
  - Every attribute of R appears as an attribute of one of the new relations.
- Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.
- e.g., Can decompose SNLRWH into SNLRH and RW.

Example Decomposition

- Decompositions should be used only when needed.
  - SNLRWH has FDs S → SNLRWH and R → W
  - Second FD causes violation of 3NF; W values repeatedly associated with R values. Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
    - i.e., we decompose SNLRWH into SNLRH and RW
- The information to be stored consists of SNLRWH tuples. If we just store the projections of these tuples onto SNLRH and RW, are there any potential problems that we should be aware of?

Problems with Decompositions

There are three potential problems to consider:

- Some queries become more expensive.
  - e.g., How much did sailor Joe earn? (salary = W*H)
- Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation! (Decomposition is not "lossless-join")
  - Fortunately, not in the SNLRWH example.
- Checking some dependencies may require joining the instances of the decomposed relations. (Decomposition is not "dependency-preserving")
  - Fortunately, not in the SNLRWH example.

Tradeoff: Must consider these issues vs. redundancy.

Decomposition into BCNF

- Consider relation R with FDs F. If X → Y violates BCNF, decompose R into R - Y and XY.
  - 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 → C, SD → P, J → S
  - To deal with SD → P, decompose into SDP, CSJDQV.
  - To deal with J → S, decompose CSJDQV into JS and CJDQV

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.
- e.g., decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (with respect to the FDs JP → C, SD → P and J → S).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - JPC tuples stored only for checking FD! (*Redundancy!*)

Decomposition into 3NF

- Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier).
- To ensure dependency preservation, one idea:
  - If $X \rightarrow Y$ is not preserved, add relation XY.
  - Problem is that XY may violate 3NF! e.g., consider the addition of CJP to `preserve' $JP \rightarrow C$. What if we also have $J \rightarrow C$?
- Refinement: Instead of the given set of FDs F, use a *minimal cover for F*.

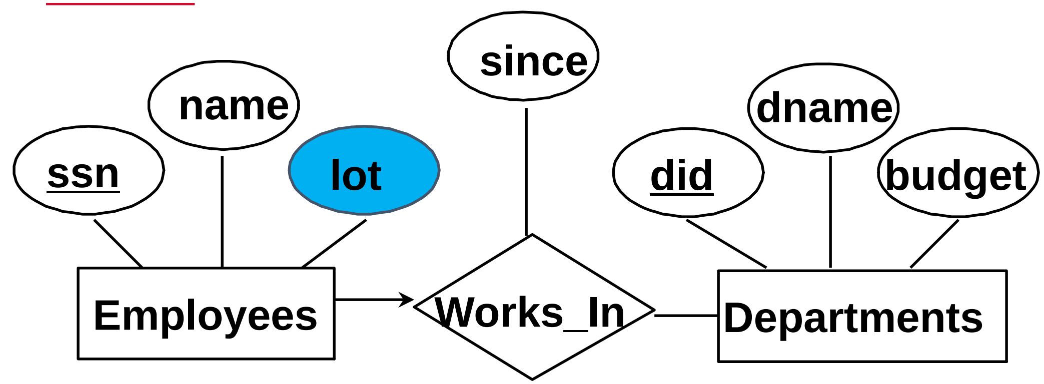
Minimal Cover for a Set of FDs

- ***Minimal cover*** G for a set of FDs F:
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.
- e.g., $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow GH$, $ACDF \rightarrow EG$ has the following minimal cover:
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$ and $EF \rightarrow H$

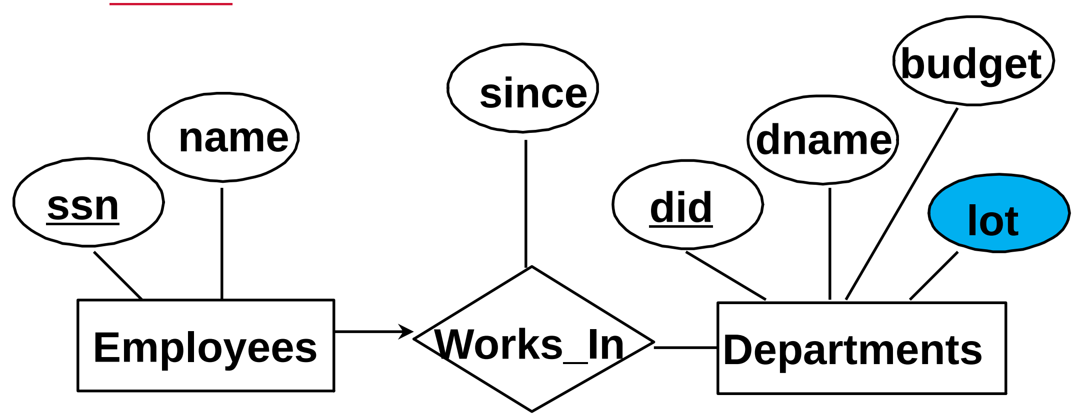
Refining an ER Diagram

- 1st diagram translated:
  Workers(S,N,L,D,S)
  Departments(D,M,B)
  - Lots associated with workers.
- Suppose all workers in a dept are assigned the same lot: D → L
- Redundancy; fixed by:
  Workers2(S,N,D,S)
  Dept_Lots(D,L)
- Can fine-tune this:
  Workers2(S,N,D,S)
  Departments(D,M,B,L)

Before:



After:



Summary of Schema Refinement

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.