Problems Redundancy Causes

- Modification anomalies:
  - **Update anomalies** – records with redundant information may not be updated at the same time
  - **Insertion anomalies** – may be no way to insert a new record
  - **Deletion anomalies** – may be no way to delete a record without deleting other information

- Extra disk storage
- Difficulties extending the database
- Inflexible querying

Modification Anomaly Examples

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Final Topics in Relational DB Design

- Attribute closure algorithm
  - Use as a test for BCNF
- Decompositions
  - Lossless-join decompositions
  - Dependency-preserving decompositions
  - Decompositions into 3NF or BCNF
- Canonical cover

Decomposing Relations into BCNF

- Suppose R is not in BCNF
- We can decompose R into two or more BCNF relations:
  - Use attribute closure to identify an FD *f* that violates BCNF
  - Use *f* to make a new relation R1
    - R1 includes all fields in *f*
    - Left side of *f* is primary key for R1
  - Modify original relation R:
    - Left side of *f* becomes foreign key to R'
    - Delete fields that are on the right side of *f*
  - Loop until all relations are in BCNF

Example

- Schema:
  - class(course_id, title, credits, sec_id, semester, year, building, room_no, capacity, time_slot_id)
- FDs:
  - course_id → title, dept_name, credits
  - building, room_no → capacity
  - course_id, sec_id, semester, year → building, room_no, time_slot_id
- Candidate keys?
  - (course_id, sec_id, semester, year) is the only candidate key
- Is schema in BCNF?
  - No – 1^{st} & 2^{nd} FDs violate BCNF
- Decomposition based on 1^{st} FD:
  - R1(<u>course_id</u>, title, dept_name, credits)
  - R(<u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, building, room_no, capacity, time_slot_id)
- Is R1 in BCNF? Yes. Is R in BCNF? No – 2^{nd} FD violates BCNF.
- Decomposition based on 2^{nd} FD:
  - R2(<u>building</u>, <u>room_no</u>, capacity)
  - R(<u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, building, room_no, time_slot_id)

25

Properties of Decompositions

- Decomposing relations is one of the tools of database design
  - First draft schema may be inefficient for various reasons
    - non-BCNF, non-3NF, too many relations, relations too complex, …
  - Take a relation R and convert into new relations R1, R2, …
  - Helps simplify complex relations, eliminate redundancy
- We would like decompositions to be:
  - **lossless-join** – we can join new relations to exactly reconstruct original relation
  - **dependency-preserving** – we can still check whether all FDs hold without doing joins

 26

Lossy Decomposition

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Lossless Decompositions

- Schema R(a, b, c)
  - Can we decompose to R1(a, b), R2(b, c)?
- In terms of relational algebra:
  - Decomposition R1(a,b), R2(b,c) is **lossless-join** if and only if $\pi_{a,b}(R) \bowtie \pi_{b,c}(R) = R$
    - Otherwise it is **lossy**
- Another test:
  - If R1 ∩ R2 is a superkey for either R1 or R2, then the decomposition to R1 and R2 is lossless-join

Example

- Schema:
  - class(course_id, title, credits, sec_id, semester, year, building, room_no, capacity, time_slot_id)
- FDs:
  - course_id → title, dept_name, credits
  - building, room_no → capacity
  - course_id, sec_id, semester, year → building, room_no, time_slot_id
- Decomposition based on 1st FD:
  - R1(course_id, title, dept_name, credits)
  - R(course_id, sec_id, semester, year, building, room_no, capacity, time_slot_id)
  - Is this lossless-join?
    - Intersection of R1 and R is course_id
    - course_id is a superkey for R1
- Decomposition based on 2nd FD:
  - R2(building, room_no, capacity)
  - R(course_id, sec_id, semester, year, building, room_no, time_slot_id)
  - Is this lossless-join?
    - Intersection of R2 and R is (building, room_no)
    - (building, room_no) is superkey for R1

Dependency Preservation

- Decomposition of R that ensures that all FDs can be checked without joining relations
- Formally:
  - F is the set of FDs on R, has closure F^+
  - Decompose R to R1, R2
  - F_{R1} is the subset of F^+ that can be checked on R1
  - F_{R2} is the subset of F^+ that can be checked on R2
  - The decomposition is dependency-preserving iff $(F_{R1} \cup F_{R2})^+ = F^+$
  - Note: there can be FDs in F that are not in F_{R1} or F_{R2}

Dependency-Preserving Test

1. Check if every FD in F holds on either R1 or R2
   - If so, the decomposition is dependency-preserving
   - If not, continue to step 2

2. Run algorithm to check dependency-preserving

```
for each FD f ∈ F:
  let result = left side of f
  do:
    for each Ri in the decomposition:
      let t = (result ∩ Ri)⁺ ∩ Ri
      result = result ∪ t
  until result is unchanged
  if result contains right side of f:
    f is preserved, continue to next FD
  else
    f is not preserved, return failure
```


Example

- Schema:
  - class(course_id, title, credits, sec_id, semester, year, building, room_no, capacity, time_slot_id)
- FDs:
  - course_id → title, dept_name, credits
  - building, room_no → capacity
  - course_id, sec_id, semester, year → building, room_no, time_slot_id
- Decomposition based on 1st FD:
  - R1(<u>course_id</u>, title, dept_name, credits)
  - R(<u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, building, room_no, capacity, time_slot_id)
  - Is this dependency-preserving?
- Decomposition based on 2nd FD:
  - R2(<u>building</u>, <u>room_no</u>, capacity)
  - R(<u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, building, room_no, time_slot_id)
  - Is this dependency-preserving?

Contracts Example

- FDs:
  - contract -> supplier, project, dept, part, qty, value
  - project, part -> contract
  - supplier, dept -> part
- Initial schema:
  - Contract(contract, supplier, project, dept, part, qty, value)
- BCNF decomposition:
  - Contract(contract, supplier, project, dept, qty, value)
  - SupplierDept(supplier, dept, part)
  - Lossless-join but not dependency preserving
    - project, part -> contract is not preserved

"Top-Down" Database Design

- Start with one giant relation with all possible attributes of all possible entities/relationships
  - All requirements/FDs can hold on this relation
  - But it is probably not good design
- Recursively decompose into BCNF relations using attribute closure and BCNF decomp alg
  - All decompositions are lossless
  - When done, all relations will be in BCNF
  - Dependencies will not necessarily be preserved

Canonical Cover

- The **canonical cover** of a set of FDs F is the smallest, simplest set of FDs needed to reconstruct the closure of F
  - Call it F_c
  - Requirements:
    1. No FD in F_c has an extraneous attribute
    2. F_c^+ is equivalent to F^+
    3. Deleting any FD from F_c will result in F_c^+ being different from F^+
- Also called the **minimal cover** of F

3NF Decomposition

- An algorithm for dependency-preserving & lossless decompositions

Given a set of functional dependencies F and a relation schema R:

```
let Fc be a canonical cover for F
let i = 0

for each FD f in Fc (f = X -> Y):
  if no existing relation has X, Y as attributes:
  then i=i+1
       define schema Ri(X, Y)

if no relation contains a candidate key for R
  then i=i+1
       define schema Ri with any candidate key for R

return (R1, R2, … Ri)
```


Contracts Example

- FDs:
  - contract -> supplier, project, dept, part, qty, value
  - project, part -> contract
  - supplier, dept -> part
- Canonical cover:
  - contract -> supplier, project, dept, qty, value
    - (*not* part—part is extraneous)
  - project, part -> contract
  - supplier, dept -> part
- Initial schema:
  - Contract(<u>contract</u>, supplier, project, dept, part, qty, value)

"Bottom-Up" Design

- Given functional dependencies F:
  - Compute canonical cover of F
  - Use 3NF "decomposition" algorithm to form relations
    - Really more of a synthesis than a decomposition
- Resulting relations will be lossless, dependency-preserving, and in 3NF
  - (many will actually be in BCNF)

