



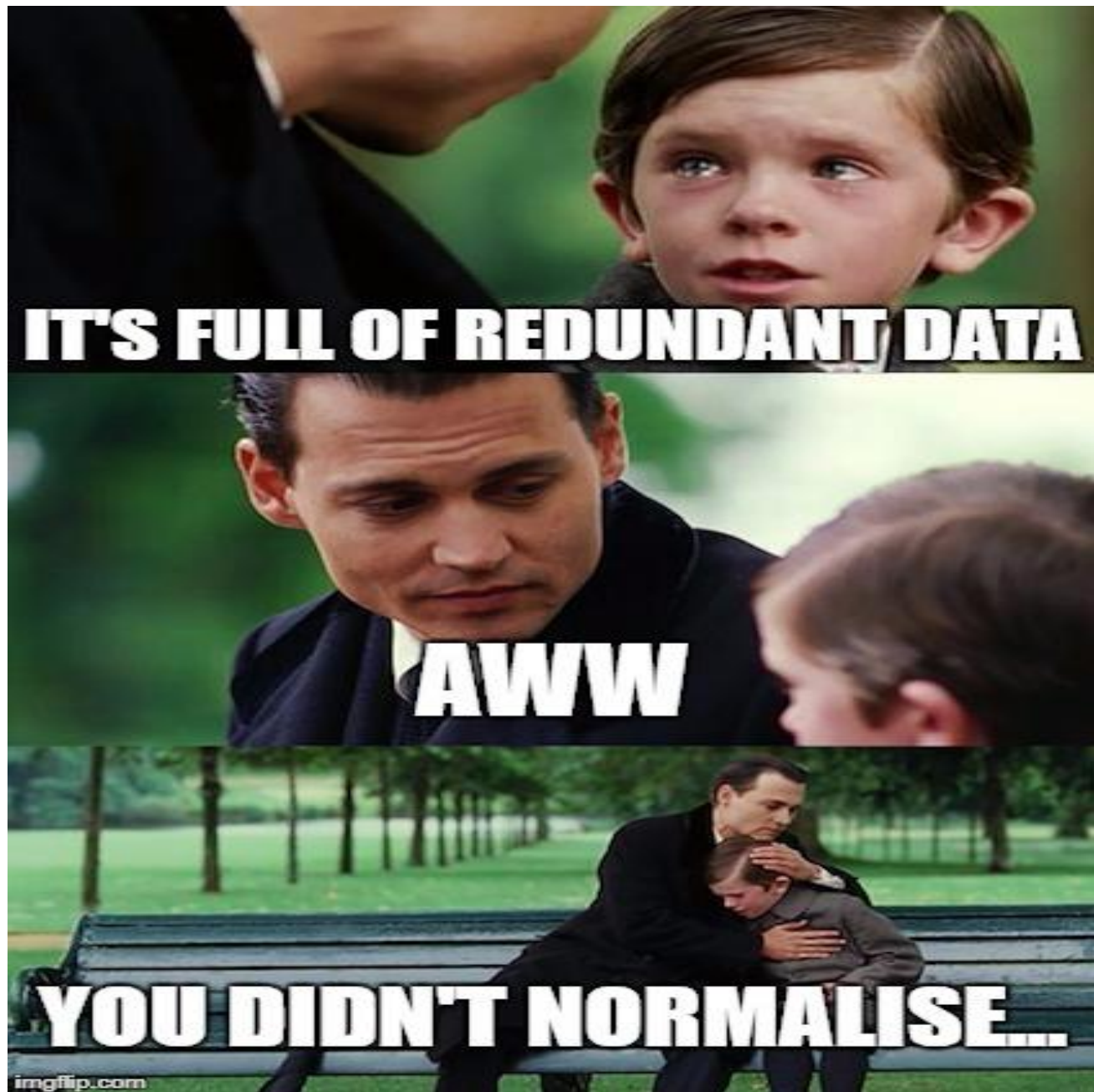
# Relational Database Design

Instructor : Nitesh Kumar Jha

[niteshjha@soa.ac.in](mailto:niteshjha@soa.ac.in)

ITER,S'O'A(DEEMED TO BE UNIVERSITY)

Sept 2018



# Boyce-Codd Normal Form

A relation schema  $R$  is in BCNF with respect to a set  $F$  of functional dependencies if for all functional dependencies in  $F^+$  of the form  $\alpha \rightarrow \beta$ , where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following holds:

- $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$ )
- $\alpha$  is a superkey for  $R$

Example schema *not* in BCNF:

*instr\_dept* (ID, *name*, *salary*, *dept\_name*, *building*, *budget* )

because  $\text{dept\_name} \rightarrow \text{building, budget}$  holds on *instr\_dept*, but *dept\_name* is not a superkey

# Decomposing a Schema into BCNF

- Suppose we have a schema  $R$  and a non-trivial dependency  $\alpha \rightarrow \beta$  causes a violation of BCNF.

We decompose  $R$  into:

- $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$
- In our example,
    - $\alpha = dept\_name$
    - $\beta = building, budget$and *inst\_dept* is replaced by
    - $(\alpha \cup \beta) = (dept\_name, building, budget)$
    - $(R - (\beta - \alpha)) = (ID, name, salary, dept\_name)$

# Testing for BCNF

- To check if a non-trivial dependency  $\alpha \rightarrow \beta$  causes a violation of BCNF
  1. compute  $\alpha^+$  (the attribute closure of  $\alpha$ , and
  2. verify that it includes all attributes of  $R$ , that is, it is a superkey of  $R$ .
- **Simplified test:** To check if a relation schema  $R$  is in BCNF, it suffices to check only the dependencies in the given set  $F$  for violation of BCNF, rather than checking all dependencies in  $F^+$ .
  - If none of the dependencies in  $F$  causes a violation of BCNF, then none of the dependencies in  $F^+$  will cause a violation of BCNF either.
- However, **simplified test using only  $F$  is incorrect when testing a relation in a decomposition of  $R$** 
  - Consider  $R = (A, B, C, D, E)$ , with  $F = \{A \rightarrow B, BC \rightarrow D\}$ 
    - ▶ Decompose  $R$  into  $R_1 = (A, B)$  and  $R_2 = (A, C, D, E)$
    - ▶ Neither of the dependencies in  $F$  contain only attributes from  $(A, C, D, E)$  so we might be misled into thinking  $R_2$  satisfies BCNF.
    - ▶ In fact, dependency  $AC \rightarrow D$  in  $F^+$  shows  $R_2$  is not in BCNF.

# Testing Decomposition for BCNF

- To check if a relation  $R_i$  in a decomposition of  $R$  is in BCNF,
  - Either test  $R_i$  for BCNF with respect to the **restriction** of  $F$  to  $R_i$  (that is, all FDs in  $F^+$  that contain only attributes from  $R_i$ )
  - or use the original set of dependencies  $F$  that hold on  $R$ , but with the following test:
    - for every set of attributes  $\alpha \subseteq R_i$ , check that  $\alpha^+$  (the attribute closure of  $\alpha$ ) either includes no attribute of  $R_i - \alpha$ , or includes all attributes of  $R_i$ .
  - ▶ If the condition is violated by some  $\alpha \rightarrow \beta$  in  $F$ , the dependency
$$\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$$
can be shown to hold on  $R_i$ , and  $R_i$  violates BCNF.
  - ▶ We use above dependency to decompose  $R_i$

# BCNF Decomposition Algorithm

```
result := {R };  
done := false;  
compute  $F^+$ ;  
while (not done) do  
  if (there is a schema  $R_i$  in result that is not in BCNF)  
    then begin  
      let  $\alpha \rightarrow \beta$  be a nontrivial functional dependency that  
        holds on  $R_i$  such that  $\alpha \rightarrow R_i$  is not in  $F^+$ ,  
        and  $\alpha \cap \beta = \emptyset$ ;  
      result := (result -  $R_i$ )  $\cup$  ( $R_i - \beta$ )  $\cup$  ( $\alpha, \beta$ );  
    end  
  else done := true;
```

Note: each  $R_i$  is in BCNF, and decomposition is lossless-join.

# Example of BCNF Decomposition

- $R = (A, B, C)$   
 $F = \{A \rightarrow B$   
 $B \rightarrow C\}$   
Key =  $\{A\}$
- $R$  is not in BCNF ( $B \rightarrow C$  but  $B$  is not superkey)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A, B)$



# Example of BCNF Decomposition

- *class (course\_id, title, dept\_name, credits, sec\_id, semester, year, building, room\_number, capacity, time\_slot\_id)*
- Functional dependencies:
  - *course\_id* → *title, dept\_name, credits*
  - *building, room\_number* → *capacity*
  - *course\_id, sec\_id, semester, year* → *building, room\_number, time\_slot\_id*
- A candidate key {*course\_id, sec\_id, semester, year*}.
- BCNF Decomposition:
  - *course\_id* → *title, dept\_name, credits* holds
    - ▶ but *course\_id* is not a superkey.
  - We replace *class* by:
    - ▶ *course(course\_id, title, dept\_name, credits)*
    - ▶ *class-1 (course\_id, sec\_id, semester, year, building, room\_number, capacity, time\_slot\_id)*

# BCNF Decomposition (Cont.)

- *course* is in BCNF
  - How do we know this?
- *building, room\_number* → *capacity* holds on *class-1*
  - but {*building, room\_number*} is not a superkey for *class-1*.
  - We replace *class-1* by:
    - ▶ *classroom* (*building, room\_number, capacity*)
    - ▶ *section* (*course\_id, sec\_id, semester, year, building, room\_number, time\_slot\_id*)
- *classroom* and *section* are in BCNF.

# BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- $R = (J, K, L)$   
 $F = \{JK \rightarrow L$   
 $L \rightarrow K\}$

Two candidate keys =  $JK$  and  $JL$

- $R$  is not in BCNF
- Any decomposition of  $R$  will fail to preserve

$$JK \rightarrow L$$

This implies that testing for  $JK \rightarrow L$  requires a join

# BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving*.
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form*.

# How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation  
*inst\_info* (*ID*, *child\_name*, *phone*)
  - where an instructor may have more than one phone and can have multiple children

<i>ID</i>	<i>Child_name</i>	<i>phone</i>
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	Willian	512-555-4321

*inst\_info*

# How good is BCNF? (Cont.)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples

(99999, David, 981-992-3443)

(99999, William, 981-992-3443)

# How good is BCNF? (Cont.)

- Therefore, it is better to decompose *inst\_info* into:

<i>inst_child</i>	<i>ID</i>	<i>child_name</i>
	99999	David
	99999	David
	99999	William
	99999	Willian

<i>inst_phone</i>	<i>ID</i>	<i>phone</i>
	99999	512-555-1234
	99999	512-555-4321
	99999	512-555-1234
	99999	512-555-4321

- This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later.

# Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.



# Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.

Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key.

*Thank You*