Universal Relational Approach

- One single, large table
- Simple ?
- Good ? or Bad? Or just Ugly?
- Normalize it!

The Normalization Process

- The process of finding good (stable) set of relations that is a faithful model of the enterprise.
- **Decomposition (top-down process)**
  - start with a universal relation
  - identify functional dependencies
  - identify key(s)
  - If necessary use decomposition to split the universal relation into a set of relations

Deriving the Keys of R from its FDs

- Let $X \subseteq R$, X^+ is its closure
- Algorithm for finding the keys
  - $X^+ = X$;
  - If $\exists$ an used $U \rightarrow V$ in FDs and $U \subseteq X^+$
    then goto step 3; else STOP;
  - $X^+ = X^+ \cup V$;
  - If $X^+ = R$
    then STOP; else goto step 2;
- A primary key is one of the minimal keys

Universal Relation: Example of BCNF Decomposition

- ***R*** = *(branch_name, branch_city, assets, customer, loan_number, amount)*
  F = *{branch_name → assets branch_city, loan_number → amount branch_name}*
  Key = *{loan_number, customer}*
- Decomposition
  - R_1 = (*branch_name, assets, branch_city)*
  - R_2 = *(branch_name, customer, loan_number, amount)*
  - R_3 = *(branch_name, loan_number, amount)*
  - R_4 = *(customer, loan_number)*
- Final decomposition: R_1, R_3, R_4

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 FD that holds on R_i
      such that $\alpha \rightarrow R_i$ is not in F^+, and $\alpha \cap \beta = \varnothing$;
        result := (*result* – R_i) $\cup$ (R_i – β) $\cup$ (α, β);
      end
    else *done* := **true;**

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

Synthesis (bottom-up process)

- Begin with attributes and combine them into related group using functional dependencies to develop a set of normalized relations.
- A synthesis algorithm was developed by Bernstein.
- Basic steps:
  - make a list of all FDs
  - groups together those with the same determinant
  - construct a relation of each group.

Synthesis… Elaborated steps

1. Make a list of all FDs.
2. Eliminate extraneous attributes in each FD.
3. Remove any redundant FDs and find a non redundant covering of the input FDs.
   - Combine FD groups with equivalent key.
4. Group together those with the same determinant.
5. Construct a relation for each group.
6. If none of the resulting relations has the same key as one of the keys of the original universal relation, we add another relation containing one of the keys, with an empty FD set.

Example: Consider the following set of FDs.

f1: A ------> B
f2: A ------> C
f3: B ------> C
f4: B ------> D
f5: D ------> B
f6: ABC ------> F

Q: Using Synthesis construct a set 3NF relations.

Solution

f1: A→B, f2: A→C } =>A→BC (union) =>f1: A→ABC, f2: ABC→F } A→F :f6'

f1: A→B, f3: B→C } =>A→C (trans) Which is f2 so f2 is redundant

f1: A→B
f6: A→F
f3: B→C
f4: B→D
f5: D→B

R1(A,B,F)
R2 (B,C,D)

	A	B	C	D	F
R1	K	K	~~U~~ K	~~U~~ K	K
R2	U	K	K	K	U

Example: Consider the following set of FDs.

f1: A ------> BC
f2: A ------> D
f3: B ------> C
f4: C ------> D
f5: DE ------> C
f6: BC ------> D

Q: Using Synthesis construct a set 3NF relations.

Design Goals

- Goal:
  - BCNF
  - Lossless join
  - Dependency preservation
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Note, SQL does not provide a direct way of specifying FDs other than superkeys.
  - Can specify FDs using assertions, but they are expensive
  - Hard to test a FD whose left hand side is not a key

Conclusions

- Disadvantages of Normal Forms
  - It is not constructive. It does not provide a way to get a good design.
  - It can only be applied after we have a schema and tell if it is good or not.
  - It provides no conceptual design. It deals directly with relations and attributes.
  - It can be applied (more or less) only to relational schemas.
- Schema Design: Conceptual Design
  - Entity-Relationship (E-R) or UML
- In practice, E-R diagrams usually lead to tables in BCNF