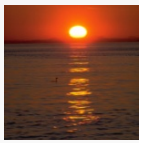
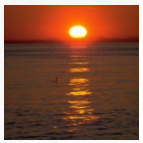
Chapter 7: Relational Database Design

Database System Concepts, 5th Ed.



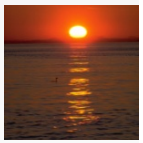
Chapter 7: Relational Database Design

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Functional Dependency Theory
- Algorithms for Functional Dependencies
- Decomposition Using Multivalued Dependencies
- More Normal Form
- Database-Design Process
- Modeling Temporal Data



The Banking Schema

- *branch* = (*branch_name*, *branch_city*, *assets*)
- *customer* = (*customer_id*, *customer_name*, *customer_street*, *customer_city*)
- *loan* = (*loan_number*, *amount*)
- *account* = (*account_number*, *balance*)
- *employee* = (*employee_id*. *employee_name*, *telephone_number*, *start_date*)
- *dependent_name* = (*employee_id*, *dname*)
- *account_branch* = (*account_number*, *branch_name*)
- *loan_branch* = (*loan_number*, *branch_name*)
- *borrower* = (*customer_id*, *loan_number*)
- *depositor* = (*customer_id*, *account_number*)
- *cust_banker* = (*customer_id*, *employee_id*, *type*)
- *works_for* = (*worker_employee_id*, *manager_employee_id*)
- *payment* = (*loan_number*, *payment_number*, *payment_date*, *payment_amount*)
- *savings_account* = (*account_number*, *interest_rate*)
- *checking_account* = (*account_number*, *overdraft_amount*)

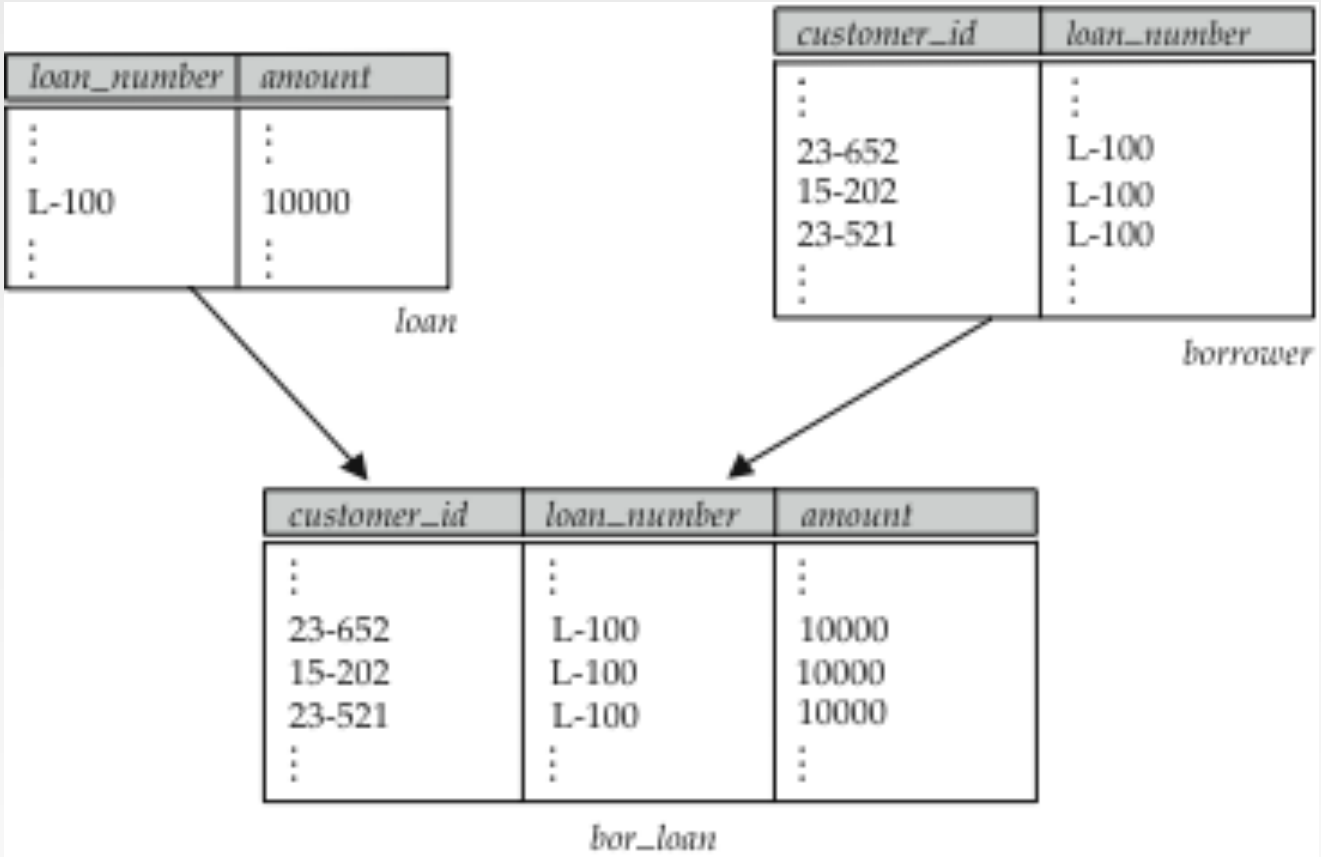


Combine Schemas?

- Suppose we combine *borrower* and *loan* to get

  bor_loan = (*customer_id*, *loan_number*, *amount*)

- Result is possible repetition of information (L-100 in example below)

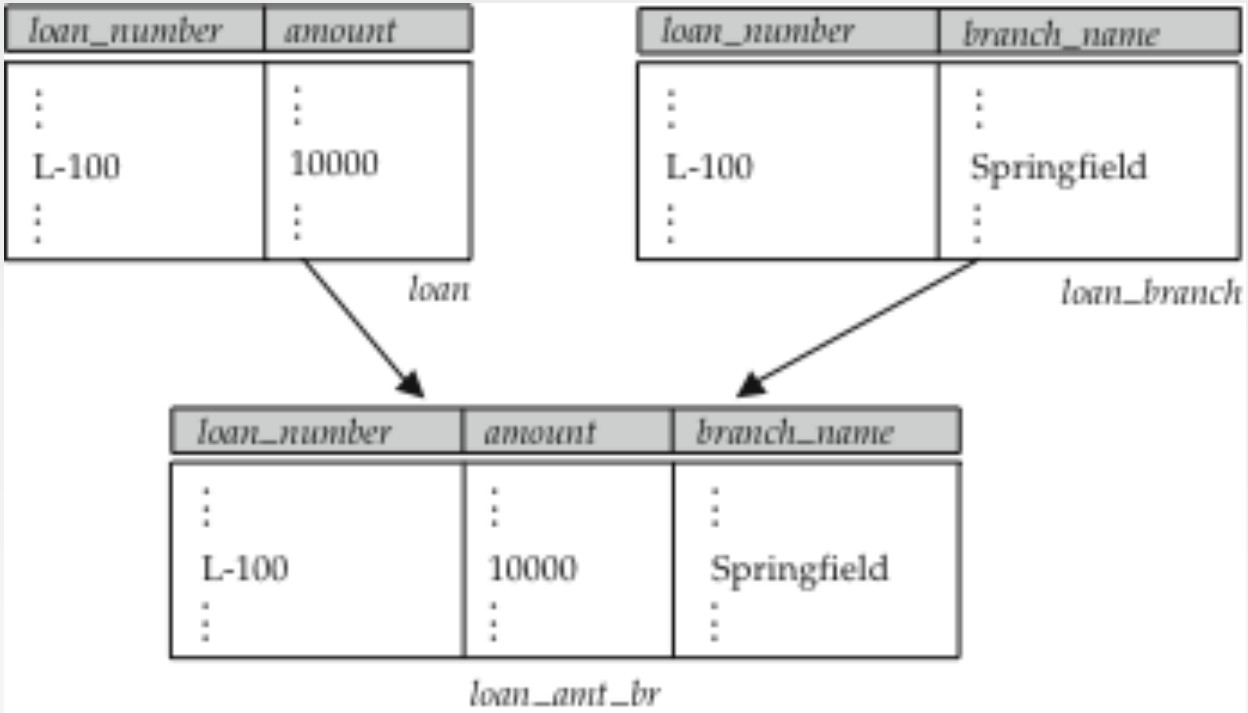


A Combined Schema Without Repetition

- Consider combining *loan_branch* and *loan*

  loan_amt_br = (*loan_number*, *amount*, *branch_name*)

- No repetition (as suggested by example below)



What About Smaller Schemas?

- Suppose we had started with *bor_loan.* How would we know to split up (**decompose**) it into *borrower* and *loan*?
- Write a rule "if there were a schema (*loan_number, amount*), then *loan_number* would be a candidate key"
- Denote as a **functional dependency**:

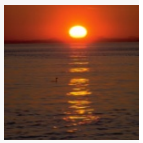
  loan_number → *amount*

- In *bor_loan*, because *loan_number* is not a candidate key, the amount of a loan may have to be repeated. This indicates the need to decompose *bor_loan*.
- Not all decompositions are good. Suppose we decompose *employee* into

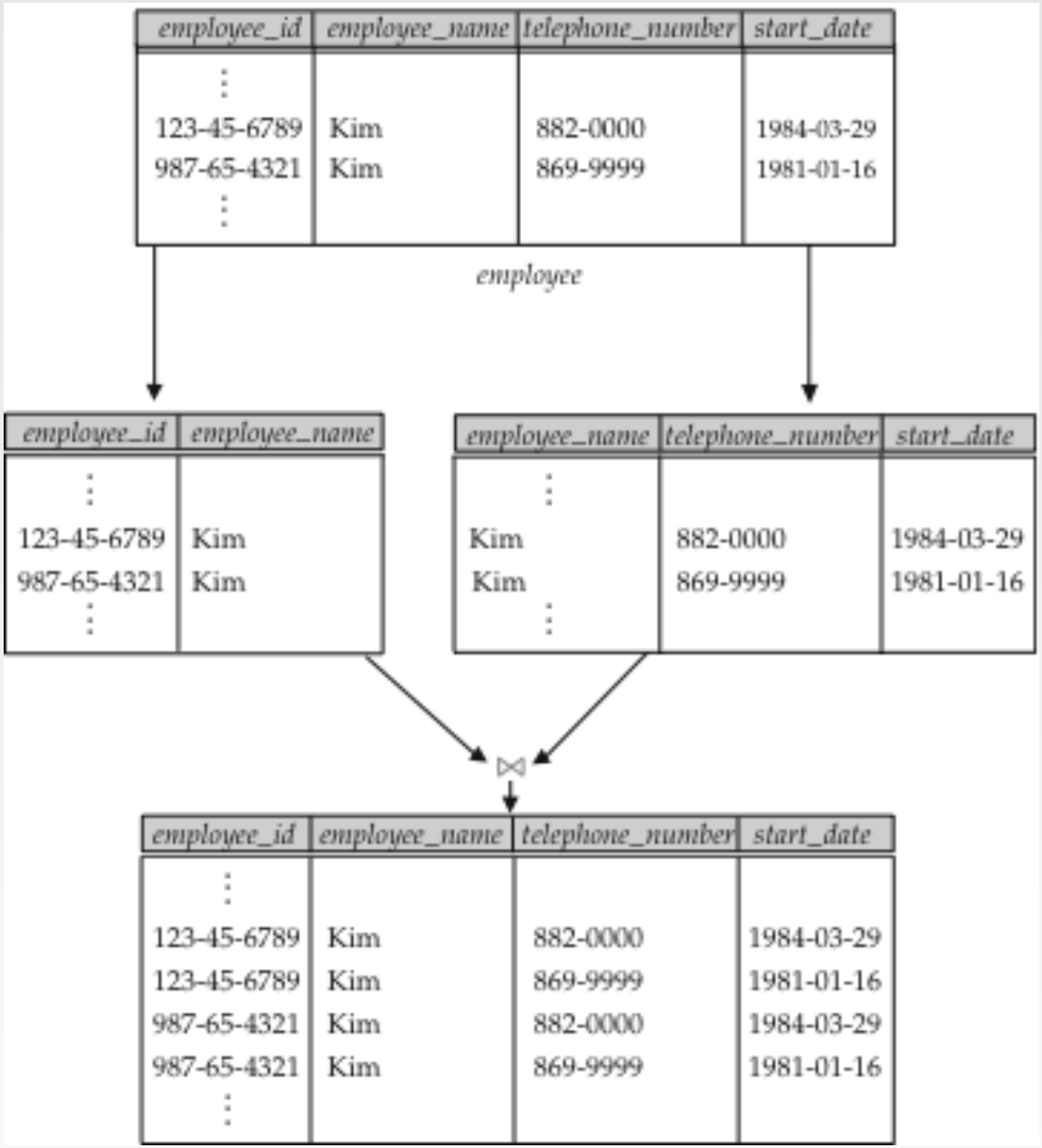
  employee1 = (*employee_id*, *employee_name*)

  employee2 = (*employee_name*, *telephone_number*, *start_date*)

- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a lossy decomposition.

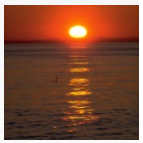


A Lossy Decomposition



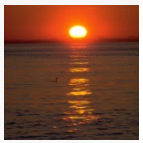
First Normal Form

- Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in first normal form if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
  - We assume all relations are in first normal form (and revisit this in Chapter 9)



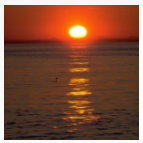
First Normal Form (Cont'd)

- Atomicity is actually a property of how the elements of the domain are used.
  - Example: Strings would normally be considered indivisible
  - Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
  - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
  - Doing so is a bad idea: leads to encoding of information in application program rather than in the database.



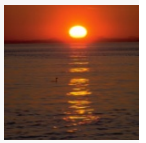
Goal — Devise a Theory for the Following

- Decide whether a particular relation R is in "good" form.
- In the case that a relation R is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies



Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key.*



Functional Dependencies (Cont.)

- Let R be a relation schema

$$\alpha \subseteq R \text{ and } \beta \subseteq R$$

- The functional dependency

$$\alpha \rightarrow \beta$$

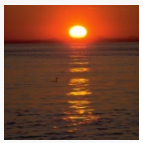
holds on R if and only if for any legal relations r(R), whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider r(A,B) with the following instance of r.

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.



Functional Dependencies (Cont.)

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

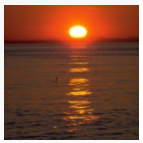
  bor_loan = (*customer_id, loan_number, amount*).

  We expect this functional dependency to hold:

  loan_number → *amount*

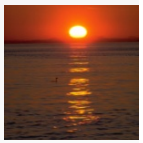
  but would not expect the following to hold:

  amount → *customer_name*



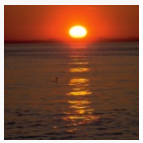
Use of Functional Dependencies

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* satisfies *F.*
  - specify constraints on the set of legal relations
    - We say that *F* holds on *R* if all legal relations on *R* satisfy the set of functional dependencies *F.*
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of *loan* may, by chance, satisfy
    amount → *customer_name.*



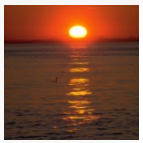
Functional Dependencies (Cont.)

- *A* functional dependency is trivial if it is satisfied by all instances of a relation
  - Example*:*
    - *customer_name, loan_number* → *customer_name*
    - *customer_name* → *customer_name*
  - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$



Closure of a Set of Functional Dependencies

- Given a set F of functional dependencies, there are certain other functional dependencies that are logically implied by F.
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of all functional dependencies logically implied by F is the *closure* of F.
- We denote the *closure* of F by F^+.
- F^+ is a superset of F.



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

$$\alpha \rightarrow \beta$$

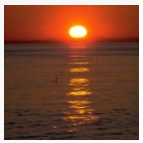
where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

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

Example schema *not* in BCNF:

bor_loan = (*customer_id, loan_number, amount*)

because *loan_number* → *amount* holds on *bor_loan* but *loan_number* 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.

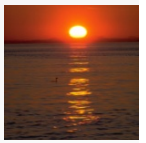
  We decompose R into:

  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$

- In our example,
  - α = *loan_number*
  - β = *amount*

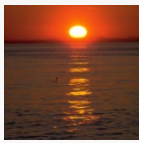
  and *bor_loan* is replaced by

  - $(\alpha \cup \beta)$ = (*loan_number, amount*)
  - $(R - (\beta - \alpha))$ = (*customer_id, loan_number*)



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.*



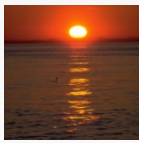
Third Normal Form

- A relation schema R is in third normal form (3NF) if for all:

  $$\alpha \rightarrow \beta \text{ in } F^+$$

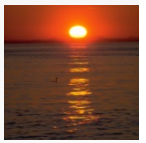
  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

    (**NOTE**: each attribute may be in a different candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).



Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies.
- Decide whether a relation scheme R is in "good" form.
- In the case that a relation scheme R is not in "good" form, decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that
  - each relation scheme is in good form
  - the decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving.

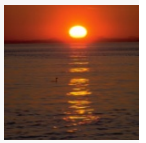


How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a database

  classes (*course, teacher, book*)

  such that (*c, t, b*) ∈ *classes* means that *t* is qualified to teach *c,* and *b* is a required textbook for *c*
- The database is supposed to list for each course the set of teachers any one of which can be the course's instructor, and the set of books, all of which are required for the course (no matter who teaches it).



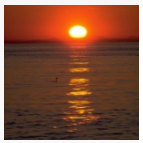
How good is BCNF? (Cont.)

course	*teacher*	*book*
database	Avi	DB Concepts
database	Avi	Ullman
database	Hank	DB Concepts
database	Hank	Ullman
database	Sudarshan	DB Concepts
database	Sudarshan	Ullman
operating systems	Avi	OS Concepts
operating systems	Avi	Stallings
operating systems	Pete	OS Concepts
operating systems	Pete	Stallings

classes

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if Marilyn is a new teacher that can teach database, two tuples need to be inserted

(database, Marilyn, DB Concepts)
(database, Marilyn, Ullman)



How good is BCNF? (Cont.)

- Therefore, it is better to decompose *classes* into:

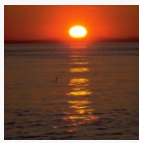
course	*teacher*
database	Avi
database	Hank
database	Sudarshan
operating systems	Avi
operating systems	Jim

teaches

course	*book*
database	DB Concepts
database	Ullman
operating systems	OS Concepts
operating systems	Shaw

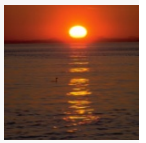
text

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



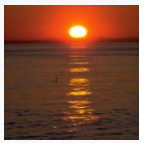
Functional-Dependency Theory

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- We then develop algorithms to generate lossless decompositions into BCNF and 3NF
- We then develop algorithms to test if a decomposition is dependency-preserving



Closure of a Set of Functional Dependencies

- Given a set *F* set of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of all functional dependencies logically implied by *F* is the *closure* of *F*.
- We denote the *closure* of *F* by F^+.
- We can find all of F^+ by applying Armstrong's Axioms:
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**
- These rules are
  - sound (generate only functional dependencies that actually hold) and
  - complete (generate all functional dependencies that hold).

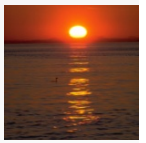


Example

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$,

      and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$,

      and then transitivity

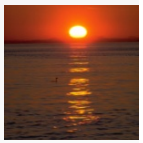


Procedure for Computing F^+

- To compute the closure of a set of functional dependencies F:

$F^+ = F$
repeat
  for each functional dependency f in F^+
    apply reflexivity and augmentation rules on f
    add the resulting functional dependencies to F^+
  for each pair of functional dependencies f_1 and f_2 in F^+
    if f_1 and f_2 can be combined using transitivity
      then add the resulting functional dependency to F^+
until F^+ does not change any further

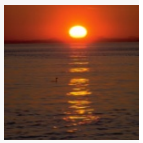
NOTE: We shall see an alternative procedure for this task later



Closure of Functional Dependencies (Cont.)

- We can further simplify manual computation of F^+ by using the following additional rules.
  - If $\alpha \rightarrow \beta$ holds *a*nd $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds **(decomposition)**
  - If $\alpha \rightarrow \beta$ holds *a*nd $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds **(pseudotransitivity)**

  The above rules can be inferred from Armstrong's axioms.

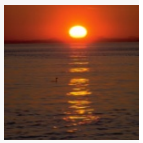


Closure of Attribute Sets

- Given a set of attributes α, define the *closure* of α under F (denoted by α^+) as the set of attributes that are functionally determined by α under F

- Algorithm to compute α^+, the closure of α under F

```
result := α;
while (changes to result) do
        for each β → γ in F do
            begin
                if β ⊆ result then  result := result ∪ γ
            end
```



Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- $(AG)^+$
  1. $result = AG$
  2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. $result = ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$
- Is AG a candidate key?
  1. Is AG a super key?
     1. Does $AG \rightarrow R$? == Is $(AG)^+ \subseteq R$
  3. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$? == Is $(A)^+ \subseteq R$
     2. Does $G \rightarrow R$? == Is $(G)^+ \subseteq R$