Chapter 16: Relational Database Design Algorithms and Further Dependencies

DESIGNING A SET OF RELATIONS

- The Approach of Relational Synthesis (Bottom-up Design):
  - Assumes that all possible functional dependencies are known.
  - First constructs a minimal set of FDs
  - Then applies algorithms that construct a target set of 3NF or BCNF relations.
  - Additional criteria may be needed to ensure the set of relations in a relational database are satisfactory.

- Goals:
  - Lossless join property (a must)
    - Algorithm 16.3 tests for general losslessness.
  - Dependency preservation property
    - Algorithm 16.5 decomposes a relation into BCNF components by sacrificing the dependency preservation.
  - Additional normal forms
    - 4NF (based on multi-valued dependencies)

1. Further Topics in FDs: Inference Rules, Equivalence, and Minimal Cover

- Functional dependencies (FDs)
  - Are used to specify formal measures of the "goodness" of relational designs
  - And keys are used to define normal forms for relations
  - Are constraints that are derived from the meaning and interrelationships of the data attributes

- A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y

- X -> Y holds if whenever two tuples have the same value for X, they must have the same value for Y
  - For any two tuples t1 and t2 in any relation instance r(R): If t1[X]=t2[X], then t1[Y]=t2[Y]

- X -> Y in R specifies a constraint on all relation instances r(R)
- Written as X -> Y; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow: ).
- FDs are derived from the real-world constraints on the attributes

Examples:

Social security number determines employee name
SSN -> ENAME
Project number determines project name and location
\( PNUMBER \rightarrow \{PNAME, PLOCATION\} \)

Employee ssn and project number determines the hours per week that the employee
works on the project
\( \{SSN, PNUMBER\} \rightarrow \text{HOURS} \)

- A FD is a property of the attributes in the schema \( R \)
- The constraint must hold on every relation instance \( r(R) \)
- If \( K \) is a key of \( R \), then \( K \) functionally determines all attributes in \( R \)
  - (since we never have two distinct tuples with \( t1[K] = t2[K] \))

1.1 Inference Rules for FDs

- Given a set of FDs \( F \), we can infer additional FDs that hold whenever the FDs in \( F \) hold
- Armstrong’s inference rules:
  - IR1. (Reflective) If \( Y \subseteq X \), then \( X \rightarrow Y \)
  - IR2. (Augmentation) If \( X \rightarrow Y \), then \( XZ \rightarrow YZ \)
    - (Notation: \( XZ \) stands for \( X \cup Z \))
  - IR3. (Transitive) If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z \)
- IR1, IR2, IR3 form a sound and complete set of inference rules
  - These are rules hold and all other rules that hold can be deduced from these

- Some additional inference rules that are useful:
  - Decomposition: If \( X \rightarrow YZ \), then \( X \rightarrow Y \) and \( X \rightarrow Z \)
  - Union: If \( X \rightarrow Y \) and \( X \rightarrow Z \), then \( X \rightarrowYZ \)
  - Pseudotransitivity: If \( X \rightarrow Y \) and \( WY \rightarrow Z \), then \( WX \rightarrow Z \)
- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

- **Closure** of a set \( F \) of FDs is the set \( F^+ \) of all FDs that can be inferred from \( F \)
- **Closure** of a set of attributes \( X \) with respect to \( F \) is the set \( X^+ \) of all attributes that are functionally determined by \( X \)
- \( X^+ \) can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in \( F \)

**Algorithm 16.1.** Determining \( X^+ \), the Closure of \( X \) under \( F \)

**Input:** A set \( F \) of FDs on a relation schema \( R \), and a set of attributes \( X \), which is a subset of \( R \).

\[
X^+ := X;
\]

repeat
  \[
  \text{old}X^+ := X^+;
  \]
  for each functional dependency \( Y \rightarrow Z \) in \( F \) do
    if \( X^+ \supseteq Y \) then \( X^+ := X^+ \cup Z \);
  \]
until \( X^+ = \text{old}X^+ \);

2
1.2. Equivalence Sets of FDs

- Two sets of FDs F and G are **equivalent** if:
  - Every FD in F can be inferred from G, and
  - Every FD in G can be inferred from F
  - Hence, F and G are equivalent if \( F^+ = G^+ \)

- **Definition (Covers):**
  - F **covers** G if every FD in G can be inferred from F
    - (i.e., if \( G^+ \subseteq F^+ \))
  - F and G are equivalent if F covers G and G covers F
  - There is an algorithm for checking equivalence of sets of FDs

1.3. Minimal Sets of FDs

- A set of FDs is **minimal** if it satisfies the following conditions:
  - Every dependency in F has a single attribute for its RHS.
  - We cannot replace any dependency \( X \rightarrow A \) in F with a dependency \( Y \rightarrow A \), where \( Y \) proper-subset-of \( X \) and still have a set of dependencies that is equivalent to F.
  - We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs
- To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal set

**Algorithm 16.2.** Finding a Minimal Cover F for a Set of Functional Dependencies E

**Input:** A set of functional dependencies E.

1. Set \( F := E \).
2. Replace each functional dependency \( X \rightarrow \{ A_1, A_2, \ldots, A_n \} \) in F by the n functional dependencies \( X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n \).
3. For each functional dependency \( X \rightarrow A \) in F
   - for each attribute \( B \) that is an element of \( X \)
     - if \( \{ F - \{ X \rightarrow A \} \} \cup \{ (X - \{B\}) \rightarrow A \} \) is equivalent to F
       - then replace \( X \rightarrow A \) with \( (X - \{B\}) \rightarrow A \) in F.
4. For each remaining functional dependency \( X \rightarrow A \) in F
   - if \( \{ F - \{ X \rightarrow A \} \} \) is equivalent to F,
     - then remove \( X \rightarrow A \) from F.
Properties of Relational Decomposition

- Universal Relation Schema:
  - A relation schema \( R = \{ A_1, A_2, \ldots, A_n \} \) that includes all the attributes of the database.
- Universal relation assumption:
  - Every attribute name is unique.

- Decomposition:
  - The process of decomposing the universal relation schema \( R \) into a set of relation schemas \( D = \{ R_1, R_2, \ldots, R_m \} \) that will become the relational database schema by using the functional dependencies.
- Attribute preservation condition:
  - Each attribute in \( R \) will appear in at least one relation schema \( R_i \) in the decomposition so that no attributes are “lost”.
- Another goal of decomposition is to have each individual relation \( R_i \) in the decomposition \( D \) be in BCNF or 3NF.
- Additional properties of decomposition are needed to prevent from generating spurious tuples (loseless; non-additive)

- Dependency Preservation Property of a Decomposition:
  - Definition: Given a set of dependencies \( F \) on \( R \), the projection of \( F \) on \( R_i \), denoted by \( \pi_{R_i}(F) \) where \( R_i \) is a subset of \( R \), is the set of dependencies \( X \rightarrow Y \) in \( F^+ \) such that the attributes in \( X \) union \( Y \) are all contained in \( R_i \).
  - Hence, the projection of \( F \) on each relation schema \( R_i \) in the decomposition \( D \) is the set of functional dependencies in \( F^+ \), the closure of \( F \), such that all their left- and right-hand-side attributes are in \( R_i \).
  - Dependency Preservation Property:
    - A decomposition \( D = \{ R_1, R_2, \ldots, R_m \} \) of \( R \) is dependency-preserving with respect to \( F \) if the union of the projections of \( F \) on each \( R_i \) in \( D \) is equivalent to \( F \); that is
      \[ ((\pi_{R_1}(F)) \cup \ldots \cup (\pi_{R_m}(F)))^+ = F^+ \]
    - (See examples in Fig 15.13a and Fig 15.12)
- Claim 1:
  - It is always possible to find a dependency-preserving decomposition \( D \) with respect to \( F \) such that each relation \( R_i \) in \( D \) is in 3NF.

**Algorithm 16.2(a).** Finding a Key \( K \) for \( R \) Given a set \( F \) of Functional Dependencies

**Input:** A relation \( R \) and a set of functional dependencies \( F \) on the attributes of \( R \).

1. Set \( K := R \).
2. For each attribute \( A \) in \( K \)
   - \{compute \( (K - A)^+ \) with respect to \( F \);
     - if \( (K - A)^+ \) contains all the attributes in \( R \), then set \( K := K - \{ A \} \} \};

2. Properties of Relational Decomposition

**Input:** A universal relation R, a decomposition D = {R1, R2, ..., Rm} of R, and a set F of functional dependencies.

**Step 1:** Create an initial matrix S with one row i for each relation Ri in D, and one column j for each attribute Aj in R.

**Step 2:** Set S(i,j):=bij for all matrix entries. (* each bij is a distinct symbol associated with indices (i,j) *).

**Step 3:** For each row i representing relation schema Ri
   {for each column j representing attribute Aj
      {if (relation Ri includes attribute Aj) then set S(i,j):= aj;};
   };
   (* each aj is a distinct symbol associated with index (j) *)

**Step 4:** Repeat the following loop until a complete loop execution results in no changes to S
   {for each functional dependency X \( \rightarrow \) Y in F
      {for all rows in S which have the same symbols in the columns corresponding to attributes in X
         {make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows:
            If any of the rows has an “a” symbol for the column, set the other rows to that same “a” symbol in the column.
            If no “a” symbol exists for the attribute in any of the rows, choose one of the “b” symbols that appear in one of the rows for the attribute and set the other rows to that same “b” symbol in the column ;};
      };
   };

**Step 5:** If a row is made up entirely of “a” symbols, then the decomposition has the lossless join property; otherwise it does not.

[Algorithm 16.4] Relational Synthesis into 3NF with Dependency Preservation (Relational Synthesis Algorithm)

**Input:** A universal relation R and a set of functional dependencies F on the attributes of R.

**Step 1:** Find a minimal cover G for F (use Algorithm 16.2);

**Step 2:** For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes \( \{X \cup \{A1\} \cup \{A2\} ... \cup \{Ak\}\}\), where X \( \rightarrow \) A1, X \( \rightarrow \) A2, ..., X \( \rightarrow \) Ak are the only dependencies in G with X as left-hand-side (X is the key of this relation);

**Step 3:** Place any remaining attributes (that have not been placed in any relation) in a single relation schema to ensure the attribute preservation property.

**Claim 3:** Every relation schema created by Algorithm 16.4 is in 3NF.
[Algorithm 16.5] Relational Decomposition into BCNF with Lossless (non-additive) join property

**Input:** A universal relation R and a set of functional dependencies F on the attributes of R.

**Step 1:** Set $D := \{R\}$;

**Step 2:** While there is a relation schema $Q$ in $D$ that is not in BCNF

- choose a relation schema $Q$ in $D$ that is not in BCNF;
- find a functional dependency $X \rightarrow Y$ in $Q$ that violates BCNF;
- replace $Q$ in $D$ by two relation schemas $(Q - Y)$ and $(X \cup Y)$;


}[Algorithm 16.6] Relational Synthesis into 3NF with Dependency Preservation and Lossless (Non-Additive) Join Property

**Input:** A universal relation R and a set of functional dependencies F on the attributes of R.

**Step 1:** Find a minimal cover $G$ for $F$ (Use Algorithm 16.2).

**Step 2:** For each left-hand-side $X$ of a functional dependency that appears in $G$,

- create a relation schema in $D$ with attributes $\{X \cup \{A_1\} \cup \{A_2\} ... \cup \{A_k\}\}$,
  where $X \rightarrow A_1, X \rightarrow A_2, ..., X \rightarrow A_k$ are the only dependencies in $G$ with $X$ as left-hand-side ($X$ is the key of this relation).

**Step 3:** If none of the relation schemas in $D$ contains a key of $R$, then create one more relation schema in $D$ that contains attributes that form a key of $R$. (Use Algorithm 16.2a to find the key of $R$)

**Step 4:** Eliminate redundant relations from the resulting set of relations. A relation $R$ is considered redundant if $R$ is a projection of another relation $S$ in the schema (i.e., $R$ is subsumed by $S$).

[Algorithm 16.2a] Finding a Key $K$ for $R$ Given a set $F$ of Functional Dependencies

**Input:** A universal relation $R$ and a set of functional dependencies $F$ on the attributes of $R$.

**Step 1:** Set $K := R$;

**Step 2:** For each attribute $A$ in $K$

- Compute $(K - A)^+$ with respect to $F$;
- If $(K - A)^+$ contains all the attributes in $R$,
  then set $K := K - \{A\}$;