Database Design

- **Purpose:** organize to achieve efficiency, ...
- **Considerations**
  - time/space tradeoff for application
  - balance application characteristics, management requirements, competing theories, ...
  - tool support and tool development based on theory
- **Approach**
  - systematic
  - transformational
  - guided by theory, but tempered by application semantics
  - adjusted by application-dependent cost analysis

Design Steps

- ORM application model
- **Model-theoretic view**
  - not efficiently organized
- If transformations and adjustments preserve information and constraints, the generated schemes and constraints are equivalent to the originals in the model-theoretic view.
- Generation of basic constraints
- ORM hypergraph
- Scheme generation
- DB schemes and constraints
  - efficiently organized
- time/space adjustments
- design transformations
Hypergraphs

- Graph = (V, E)
  - V = set of vertices
  - E = set of edges (two vertices; one for unordered loops)
    - e ∈ E can be unordered: e = {x, y}
    - e ∈ E can be ordered: e = (x, y)

- Hypergraph = (V, E)
  - V = set of vertices
  - E = set of edges (can have more than two vertices)
    - e ∈ E can be unordered: e = {x₁, x₂, ..., xₙ}
    - e ∈ E can be ordered: e = (x₁, ..., xₘ; y₁, ..., yₖ) = (x₁, ..., xₘ -- y₁, ..., yₖ)

ORM Hypergraphs

- A regular hypergraph with:
  - object sets as vertices
  - relationship sets as edges

- Plus:
  - Generalization/Specialization
  - Optional-participation markers
  - General constraints

- Example:
Conversion to ORM Hypergraph

\[
\begin{align*}
\forall x \in (H(x) \rightarrow ^4 \{y, z, w\}, (D(y) \rightarrow (J(w))))
\end{align*}
\]

Functional Dependencies (FDs)

Let \( r(R) \) be a relation and let \( t \in r \), then the \textit{restriction} of \( t \) to \( X \subseteq R \), written \( t[X] \), is the projection of \( t \) onto \( X \).

\[
\begin{align*}
R(A, B, C) & \quad BC \subseteq ABC \\
t = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & 5 \\ 2 & 3 & 5 \\ 4 & 4 & 5 \end{pmatrix} \\
\text{t}[BC] &= \{(B:2), (C:5)\}
\end{align*}
\]

Let \( R \) be a relation scheme and let \( X \subseteq R \) and \( Y \subseteq R \).

\( X \rightarrow Y \) is a \textit{functional dependency} or \textit{FD}. A relation \( r(R) \) \textit{satisfies} the FD \( X \rightarrow Y \) (or \( X \rightarrow Y \) \textit{holds}) if for any two tuples \( t_1 \) and \( t_2 \) in \( r \), \( t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y] \); alternatively (and equivalently) if for every (sub)tuples \( x \) in \( \pi_X r \), \( |\sigma_{x \subseteq x} \Pi_{X \subseteq X} r| = 1 \).

\[
\begin{align*}
A \rightarrow C & \quad B \rightarrow C \\
AB \rightarrow C & \quad AC \rightarrow B
\end{align*}
\]
Directed Edges in ORM Hypergraphs are Functional Dependencies

Generated Constraints:
- Participation Constraint: \( \forall x (A(x) \rightarrow \exists y (A(x) \land B(y))) \)
- Referential-Integrity Constraint: \( \forall x \forall y (A(x) \land B(y)) \rightarrow A(x) \land B(y) \)

Proof that these constraints imply that \( ArB \) satisfies \( A \rightarrow B \).

Recall: \( A \rightarrow B \) if for every pair of tuples \( t_1 \) and \( t_2 \) in \( ArB \),
\( t_1[A] = t_2[A] \rightarrow t_1[B] = t_2[B] \).

Proof: Let \( t_1 \) and \( t_2 \) be tuples in \( ArB \) such that \( t_1[A] = t_2[A] \).
Since \( t_1 \in ArB \), by the referential-integrity constraint, \( t_1[A] \in A \).
Thus, by the participation constraint \( \exists y (A(x) \land B(y)) \) and thus there is exactly one \( y \) such that \( A(x) \land B(y) \). Hence, \( t_1[B] = t_2[B] \).

Proofs for other cases are similar (see Appendix C).

Motivational Example

Reduction Transformation:
Information Preserving
Constraint Preserving

\( \forall x \forall y z (Worker(x) \text{ works for Manager(y) as of Start Date(z)}) \)
\( \lor (Worker(x) \land Manager(y) \land Start Date(z)) \)
\( \lor (Worker(x) \land \exists y \exists z \forall x (Worker(x) \text{ works for Manager(y) as of Start Date(z)))} \)
FD Implication

Let \( r(R) \) and let \( F \) be a set of FDs over \( R \). Then \( r \) *satisfies* \( F \) if each FD in \( F \) holds for \( r \).

\( F \) may imply that other FDs also hold. \( F \) *implies* \( X \rightarrow Y \) (sometimes written \( F = X \rightarrow Y \) or \( F \Rightarrow X \rightarrow Y \)) if \( X \rightarrow Y \) holds for every relation that satisfies \( F \).

\[
\begin{array}{cccc}
  r = A & B & C & D \\
  1 & 2 & 3 & 4 \\
  5 & 6 & 7 & 8 \\
  5 & 6 & 7 & 9 \\
  0 & 1 & 2 & 3 \\
\end{array}
\]

\[
F = \{ A \rightarrow B, B \rightarrow C \}
\]

\( r \) satisfies \( F \).

\( F \) implies \( A \rightarrow C \).

Proof:
1. Let \( s[A] = t[A] \).
2. \( s[A] = t[A] \rightarrow s[B] = t[B] \). given, \( A \rightarrow B \)
3. \( s[B] = t[B] \). 1 & 2, modus ponens
4. \( s[B] = t[B] \rightarrow s[C] = t[C] \). given, \( B \rightarrow C \)
5. \( s[C] = t[C] \). 3 & 4, modus ponens

\[ F^+ \]

Let \( S \) be a set of attributes. If \( F \) is a set of FDs over \( S \), the set of all FDs implied by \( F \) is called the *closure* of \( F \), denoted \( F^+ \).

When we assert an FD \( X \rightarrow Y \), we mean \( X \rightarrow Y \in F^+ \).

Rules for computing \( F^+ \):

* (trivial implication) \( Y \subseteq X \Rightarrow X \rightarrow Y \)
  
  e.g., Name City \( \rightarrow \) Name

* (accumulation) \( X \rightarrow Y, W \rightarrow Z, W \subseteq Y \Rightarrow X \rightarrow YZ \)
  
  e.g., GuestNr \( \rightarrow \) Name City, Name \( \rightarrow \) Room \( \Rightarrow \) GuestNr \( \rightarrow \) Name City Room

* (projection) \( X \rightarrow Y, Z \subseteq Y \Rightarrow X \rightarrow Z \)
  
  e.g., GuestNr \( \rightarrow \) Name City Room \( \Rightarrow \) GuestNr \( \rightarrow \) Room

We compute \( F^+ \) by a least-fixed-point process.
Sound and Complete Rules

The implication rules for F+ are:
- sound: the derived FDs hold for any relation satisfying F;
- complete: repeated application of the rules derives all implied FDs.

Proof of Soundness.
(Y ⊆ X → X → Y); Let s[X] = t[X]. Then since Y ⊆ X, s[Y] = t[Y].
(X → Y, W → Z, W ⊆ Y → X → YZ); Let s[X] = t[X]. Then since X → Y, s[Y] = t[Y].
Then, since W ⊆ Y, s[W] = t[W] and since W → Z, s[Z] = t[Z]. Now, since
s[Y] = t[Y] and s[Z] = t[Z], s[YZ] = t[YZ].
(X → Y, Z ⊆ Y → X → Z); Let s[X] = t[X]. Then since X → Y, s[Y] = t[Y], and since
Z ⊆ Y, s[Z] = t[Z].

Proof of Completeness (Appendix C).

Example of F+

If the set of attributes is ABC and the set of FDs F = {A → B, B → C},
then F+ = {A → B, B → C

A → A, B → B, C → C, AB → AB, AC → AC, BC → BC, ABC → ABC,
AB → A, AB → B, AC → A, AC → C, BC → B, BC → C,
ABC → A, ABC → B, ABC → C, ABC → AB, ABC → AC, ABC → BC,

A → BC, A → C, A → AB, A → AC, A → ABC,
B → BC, AB → C, AB → AC, AB → BC, AB → ABC,
AC → B, AC → AB, AC → BC, AC → ABC}
Additional FD-Implication Rules

(augmentation) \( X \rightarrow Y \rightarrow XZ \rightarrow YZ \)
Room \( \rightarrow \) Cost \( \Rightarrow \) Room NrDays \( \rightarrow \) Cost NrDays

(transitivity) \( X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z \)
GuestNr \( \rightarrow \) Name, Name \( \rightarrow \) Room \( \Rightarrow \) GuestNr \( \rightarrow \) Room

(union) \( X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ \)
Room \( \rightarrow \) NrBeds, Room \( \rightarrow \) Cost \( \Rightarrow \) Room \( \rightarrow \) NrBeds Cost

Checking for \( X \rightarrow Y \in F^+ \)

- Generate \( F^+ \) and see if \( X \rightarrow Y \) is present (expensive)
- Derive \( X \rightarrow Y \) from \( F \), or determine that it’s not derivable
  - Derivation sequence for \( X \rightarrow Y \): sequence of FDs
    - each FD is given in \( F \) or follows by a sound derivation rule
    - \( X \rightarrow Y \) is the last FD in the sequence
  - Examples:

    \( R = ABCD, F = \{A \rightarrow B, B \rightarrow C\}, AD \rightarrow C \in F^+, BC \rightarrow A \in F^? \)

    1. \( A \rightarrow B \) given
    2. \( B \rightarrow C \) given
    3. \( A \rightarrow C \) transitivity, 1 & 2
    4. \( AD \rightarrow CD \) augmentation, 3
    5. \( AD \rightarrow C \) projection, 4

    1. \( BC \rightarrow B \) trivial implication
    2. \( B \rightarrow C \) given
    3. \( BC \rightarrow C \) transitivity, 1 & 2
    4. \( \ldots \)

    How do we know we cannot derive \( BC \rightarrow A \)?
TAP Derivation Sequence

- A particular derivation sequence always works!
  - List the given FDs
  - T: Trivial Implication
  - A: Accumulation (repeated zero or more times)
  - P: Projection (if needed)
- Examples:

  \[ R = ABCD, \; F = \{A \rightarrow B, \; B \rightarrow C\}, \; AD \rightarrow C \subseteq F^+, \; BC \rightarrow A \subseteq F^+ \]

  1. \( A \rightarrow B \) given
  2. \( B \rightarrow C \) given
  3. \( AD \rightarrow AD \) T
  4. \( AD \rightarrow ABD \) A
  5. \( AD \rightarrow ABCD \) A
  6. \( AD \rightarrow C \) P

  How do we know we cannot derive \( BC \rightarrow A \)?
  - Accumulation yields nothing more, and
  - Projection cannot yield \( A \) on the rhs, and
  - \( BC \rightarrow A \subseteq F^+ \) if there is a TAP derivation sequence for \( BC \rightarrow A \).

\[ X^+ - Closure \; of \; a \; Set \; of \; Attributes \]

- \( X^+ \) = maximal accumulation in a TAP derivation sequence starting with \( X \).
- Algorithm for \( X^+ \) given a set of FDs \( F \):
  1. Start with \( X^+ = X \).
  2. If \( Y \rightarrow Z \in F \) and \( Y \subseteq X^+ \), \( X^+ \) becomes \( X^* \).
  3. Repeat 2 until no more changes to \( X^+ \) (least fixed point).
- Examples:

  \[ R = ABCD, \; F = \{A \rightarrow B, \; B \rightarrow C\} \]

  \[ AD^+ = ABCD \]
  \[ BC^+ = BC \]
  \[ BD^+ = BCD \]
  \[ D^+ = D \]
  \[ A^+ = ABC \]
$X \rightarrow Y \in F^+ \text{ iff } Y \subseteq X^+$

- Significant observation!
  - $X \rightarrow Y \in F^+$ looks like a problem requiring exponential time
  - BUT has a polynomial-time solution (linear with well-chosen data structures)
- This is an example of the essence of good computer science.

---

$X^+$ and Hypergraph Reachability

To test $X \rightarrow Y \in F^+$, mark the vertices in $X$ and see if the vertices in $Y$ are reachable following directed edges.

```
A → D \in F^+? Yes
A → G \in F^+? No
E → CD \in F^+? Yes
A → BH \in F^+? No
AE → HG \in F^+? Yes
...```

---
Reduction Transformations

- Discard “stuff” in model-theoretic view that we don’t need
  - Discard redundant relations (= redundant hypergraph edges)
  - Reduce relations (= reducible hypergraph edges)
  - Discard implied constraints
- Example

![Diagram of a hypergraph showing relationships between Room, Guest, and Name]

Observe:
1. \(\text{names}_\text{guest}_\text{occupying} = \pi_{\text{Room}, \text{Name}}(\text{occupies} \times \text{has})\)
2. \((\text{Room} \rightarrow \text{Guest}, \text{Guest} \rightarrow \text{Name}) \rightarrow \text{Room} \rightarrow \text{Name}\)

Information Preserving Transformations

A transformation from \(M\) to \(M'\) preserves information if for any valid interpretation there is a procedure \(P\) that computes \(M\) from \(M'\).

For our example, \(P\) is:

- \(\text{Room} = \text{Room}\)
- \(\text{Guest} = \text{Guest}\)
- \(\text{Name} = \text{Name}\)
- \(\text{occupies} = \text{occupies}\)
- \(\text{has} = \text{has}\)
- \(\text{names}_\text{guest}_\text{occupying} = \pi_{\text{Room}, \text{Name}}(\text{occupies} \times \text{has})\)
Constraint Preserving Transformations

A transformation from M to M' preserves constraints if C', the constraints of M' (along with the constraints implied by the transformations P and P'), implies C, the constraints of M.

For our example: C' \Rightarrow C

Constraints of M:  C = \{Room \rightarrow Guest, Guest \rightarrow Name, Room \rightarrow Name, \\
\forall x \forall y (Name(x) names guest occupying Room(y) \Rightarrow Name(x) \wedge Room(y)), \\
\ldots\}

Constraints of M', plus the constraints implied by P and P': C' = \{
Room \rightarrow Guest, Guest \rightarrow Name, \\
\forall x \forall y (Guest(x) occupies Room(y) \Rightarrow Guest(x) \wedge Room(y)), \\
\forall x \forall y (Guest(x) has Name(y) \Rightarrow Guest(x) \wedge Name(y)). \\
\ldots, \\
\forall x \forall y (Name(x) names guest occupying Room(y) \iff \\
\exists z (Guest(z) has Name(x) \wedge Guest(z) occupies Room(y)))
\}

Equivalence for Reduction Transformations

- The backward transformations (i.e., the ones just discussed) are the hard ones to guarantee
- The forward transformations are easy to guarantee
  - There is a “trivial” transformation P' that computes M' from M – just delete the discarded part.
  - The constraints of M imply the constraints of M' vacuously – every constraint in M' is a constraint in M.
- Reduction transformations are actually equivalence transformations.
FD Equivalence

- Two sets of FDs F & G are *equivalent*, written \( F \equiv G \), if F implies each FD in G and conversely.

\[
F = \{ A \rightarrow B, AB \rightarrow C, C \rightarrow D \} = G = \{ A \rightarrow BC, C \rightarrow D, A \rightarrow D \}
\]

In F, \( A^+ = AB \rightarrow C \), \( AB \rightarrow C \), and \( C \rightarrow D \).

In G, \( A^+ = ABCD \rightarrow A \rightarrow BC \), \( AB \rightarrow C \), and \( C^+ = CD \rightarrow C \rightarrow D \).

- If \( F \equiv G \), then \( F^+ = G^+ \).

\[
(G^+ \subseteq F^+ \land F^+ \subseteq G^+) \Rightarrow F^+ = G^+
\]

Semantics

\( q = \pi_{\text{Room}, \text{Name}}(r \times s) \) ?? 

Case 1: \( q \) is name of guest staying in room.

Case 2: \( q \) is name of room.
Techniques for Checking Semantics

- Relevant-Edge Sets
- Project-Join or Universal-Relation Test
- Outer-Join/FD Test
- Congruency Test

A search for Semantic Equivalence

Relevant Edge Sets

- Ignore edges not used in derivations.
- Multiple edge sets are possible.
- A backtracking algorithm can generate all relevant edge sets.
- Example: for q below
  - Ignore t
  - Consider: \( \{(p), \{r, s\}\} \)
Universal Relation

Universal Relation: lossless join over all relations (for all valid interpretations)

If \( r_1(R_1), \ldots, r_n(R_n) \) and \( u = r_1 \times \cdots \times r_n \), then \( \pi_R u = r_i \).

Case 1.

\[
\begin{array}{cccc}
R & \text{Guest} & \text{Name} \\
1 & 101 & Smith \\
3 & 103 & Carter \\
\end{array}
\]

\[
\begin{array}{cccc}
S & \text{Guest} & \text{Name} \\
1 & 101 & Smith \\
3 & 103 & Carter \\
\end{array}
\]

\[
\begin{array}{cccc}
Q & \text{Room} & \text{Name} \\
1 & 101 & Smith \\
3 & 103 & Carter \\
\end{array}
\]

\[
\begin{array}{cccc}
R & | \times | S & | \times | Q \\
1 & 101 & 101 & Smith \\
3 & 103 & 103 & Carter \\
\end{array}
\]

Case 2.

\[
\begin{array}{cccc}
R & \text{Guest} & \text{Name} \\
1 & 101 & Smith \\
3 & 103 & Carter \\
\end{array}
\]

\[
\begin{array}{cccc}
S & \text{Guest} & \text{Name} \\
1 & 101 & Smith \\
3 & 103 & Carter \\
\end{array}
\]

\[
\begin{array}{cccc}
Q & \text{Room} & \text{Name} \\
1 & 101 & Smith \\
3 & 103 & Carter \\
\end{array}
\]

\[
\begin{array}{cccc}
R & | \times | S & | \times | Q \\
1 & 101 & 101 & Smith \\
3 & 103 & 103 & Carter \\
\end{array}
\]

Observe that the FD \( \text{Room} \rightarrow \text{Name} \) fails.

In general, we can ask, “Do we always get to the same place.”

Outer Join

Instead of discarding “dangling tuples” (tuples that do not join), pad them with nulls and add them to the result.

\[
\begin{array}{cccc}
R & \text{Guest} & \text{Name} & \text{Room} & \text{Name} \\
1 & 101 & 101 & Smith & Kennedy \\
3 & 103 & 103 & Carter & Carter \\
\end{array}
\]

\[
\begin{array}{cccc}
S & \text{Guest} & \text{Name} \\
3 & 103 & Carter \\
\_ & 101 & Smith \\
1 & \_ & Kennedy \\
\end{array}
\]

\[
\begin{array}{cccc}
R & \text{Guest} & \text{Name} \\
3 & 103 & Carter \\
\_ & 101 & Smith \\
1 & \_ & Kennedy \\
\end{array}
\]

Observe that the FD \( \text{Room} \rightarrow \text{Name} \) fails.

In general, we can ask, “Do we always get to the same place.”
Congruency

- If the common properties of the objects in an object set $S$ coincide with the properties explicitly defined for $S$, $S$ is *congruent*; otherwise $S$ is *incongruent*.
- For ORM application models:
  - An object set $E$ is an *explicitly defined property* for an object set $S$ if $E$ is connected by a relationship set to $S$ or to any generalizations of $S$.
  - An object set $C$ is a *common property* for an object set $S$ if for any valid interpretation, all objects in $S$ connect to one or more objects in $C$.
- For ORM hypergraphs, an object set is incongruent if it has a relationship set with an optional connection; otherwise it is congruent.

Congruency – Same Semantics

Case 1: $q$ is name of guest staying in room.
Case 2: q is name of room.

Incongruent

Congruent

Semantic Equivalence

- If a subORM diagram $S$ (a set of object and relationship sets constituting a valid diagram) is congruent and has a universal relation for all valid interpretations, semantic equivalence holds over $S$.
- Various ways to view this.
  - Fundamental idea: no conflicting semantics.
  - URA (Universal-Relation Assumption)
    - join-project test
    - outer-join/FD-paths test
  - URSA (Universal Relation-Scheme Assumption)
    - congruency test
    - all attributes have one and only one meaning
    - roles are introduced to resolve multiple meanings
Union-Covered Isolated Root Generalization

Partition-Covered Isolated Root Generalization

\( \forall x (\text{Guest Phone}(x) \rightarrow \neg \text{Room Phone}(x)) \)
\( \forall x (\text{Room Phone}(x) \rightarrow \neg \text{Guest Phone}(x)) \)
Head-Reduction Test

- **Procedure**
  - Remove proposed head component.
  - Mark tails & run closure.
  - If head object set marked, "yes"; else "no".

- **Observations**
  - No head reduction is possible unless multiple arrow heads initially point at an object set.
  - We must, of course, check for semantic equivalence.

---

Head Reduction – Case 1
(One Tail or Multiple Heads)

- **Reduction:**
  - For multiple heads, discard the tested head.
  - For one tail with one head, discard the edge.

- **Preserves information**
  - Join over the relevant edge set and project on object sets of edge.
  \[ \pi_{ACD}(AB \times BC \times AD) \]

- **Preserves constraints**
  - \( A \rightarrow B, B \rightarrow C, A \rightarrow D \Rightarrow A 
  \rightarrow CD \)
  - Referential integrity and other constraints (trivially) OK
Head Reduction – Case 2
(Multiple Tails and Single Head)

- Reduction:
  - Discard the head component.
  - Keep tails connected.

- Preserves information
  \[ \alpha_{ABC}(AB | | AD | | DC) \]
  - Not enough (no B): \[ \alpha_{AC}(AD | | DC) \]
  - Not correct: \[ \alpha_{ABC}(AD | | DC | | BE) \]

- Preserves constraints: A → D, D → C → AB → C

Tail-Reduction Test

- Procedure
  - Mark all tails of an edge except the tail proposed for removal. (Be sure to keep the unmarked (questioned) tail component for the test.)
  - Run closure.
  - If head object sets marked, “yes”; else “no”.

- Observations
  - No tail reduction is possible unless the edge has multiple tails.
  - We must, of course, check for semantic equivalence.
Tail Reduction – Case 1
(Edge Used in Closure)

- Reduction: Discard the tail component.
- Preserves information
  - Join over relevant edge set and project on object sets of adjusted edge.
  \[
  \bar{\alpha} \rightarrow \pi_{ABC}(AF \times FB \times AC) \quad \begin{array}{cccc|cccc}
  1 & 3 & 5 & 1 & 5 & 1 & 1 & 3 \\
  2 & 4 & 6 & 2 & 6 & 2 & 2 & 4 \\
  5 & 1 & 1 & 3 & 5
  \end{array}
  \]
- Preserves constraints
  - \( A \rightarrow C \Rightarrow AB \rightarrow C \)
  - (note: forward transformation OK) \( A \rightarrow F, F \rightarrow B, AB \rightarrow C \Rightarrow A \rightarrow C \)
  - Referential integrity and other constraints (trivially) OK

Tail Reduction – Case 2
(Edge Not Used in Closure)

- Reduction:
  - Discard the tail component.
  - Add connection among tails.
- Preserves information
  \[
  \bar{\alpha} \rightarrow \pi_{ABC}(AB \times AC \times AD \times DC) \quad \begin{array}{ccc|ccc|ccc}
  A & D & C & A & D & C & B & E \\
  1 & 1 & 1 & 1 & 4 & 4 & 1 & 6 \\
  2 & 2 & 2 & 2 & 5 & 5 & 2 & 7
  \end{array}
  \]
- Not enough (no B):
  \[
  \bar{\alpha} \rightarrow \pi_{AC}(AC \times AD \times DC) \quad \begin{array}{ccc|ccc|ccc}
  A & C & B \\
  1 & 4 & 1 & 1 & 1 & 1
  \end{array}
  \]
- Not correct:
  \[
  \bar{\alpha} \rightarrow \pi_{AC}(AC \times AD \times DC \times BE) \quad \begin{array}{ccc|ccc|ccc}
  A & C & B \\
  2 & 2 & 1 & 2 & 2 & 2
  \end{array}
  \]
- Preserves constraints:
  - \( A \rightarrow C \Rightarrow AB \rightarrow C \)
  - \( A \rightarrow D, D \rightarrow C \Rightarrow A \rightarrow C \)
FD Equivalence Classes

- $X \equiv Y$ for a set of FDs $F$ if $X \rightarrow Y \in F^+$ and $Y \rightarrow X \in F^+$.
  - $F = \{A \rightarrow BC, BC \rightarrow D, D \rightarrow A, A \rightarrow D, BC \rightarrow E\}$
  - $A \equiv D, A \equiv BC, BC \equiv D$, but not $BC \equiv E$

$\equiv$ is an equivalence relation, with equivalence classes.
  - reflexive, symmetric, transitive
  - $\{A, BC, D\}$

- trivial equivalence classes: $\{B\}, \{ABC\}$
- nontrivial, minimal-set equivalence class
  - nontrivial & no composite set can be reduced
  - not: $\{AB, BC, D\}$
  - not: $\{B\}$

Circularly Linked Equivalence Class
Minimally Consolidated Equivalence Class

Lexicalization
Chapter 9 - 45

Composite Lexicalization

Chapter 9 - 46

1-1 Nonlexical Object Sets
Further Head Reductions

- Some head reductions are possible only after linking equivalence classes circularly.
- Figure 9.20 gives an example.
  - 9.20a: no head reductions possible
  - 9.20b: circular link added
  - 9.20c: head reduction possible

Redundant NonFD Relationship Sets

- Looks redundant.
- Is redundant if semantic equivalence holds.
- Cycles always exist when there are redundant nonFD relationship sets.
- Removal preserves information.
- Removal preserves constraints.
Information Preservation – Sufficient for NonFD Edge Reduction

Transformable if:

\[ AD = \pi_{AB}(AB \times BC \times CD) \]

Note that the cycles we need have nothing to do with the direction of edges.

Cycles must include the entire hyperedge. We cannot remove ADE because we cannot recover the connections to objects in E.

Head/Tail Reductions Can Yield a Redundant NonFD Relationship Set

Chapter 9 - 49

Chapter 9 - 50
N-ary Relationship-Set Reduction

\[ \text{Guest} \rightarrow \text{Date} \rightarrow \text{Room} \rightarrow \text{View} \]

<table>
<thead>
<tr>
<th>Guest</th>
<th>Date</th>
<th>Room</th>
<th>View</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>10 May</td>
<td>R1</td>
<td>Sea</td>
</tr>
<tr>
<td>G1</td>
<td>10 May</td>
<td>R1</td>
<td>Forest</td>
</tr>
<tr>
<td>G1</td>
<td>15 May</td>
<td>R2</td>
<td>Forest</td>
</tr>
<tr>
<td>G2</td>
<td>15 May</td>
<td>R1</td>
<td>Sea</td>
</tr>
<tr>
<td>G2</td>
<td>15 May</td>
<td>R1</td>
<td>Forest</td>
</tr>
</tbody>
</table>

Non-reducible N-ary Edge

\[ r = \text{Guest} \rightarrow \text{Date} \rightarrow \text{Room} \]

\[ p = \text{Guest} \rightarrow \text{Date} \]

\[ q = \text{Date} \rightarrow \text{Room} \]

\[ s = \text{Guest} \rightarrow \text{Room} \]

\[ r = p \rightarrow q \rightarrow s \]

\[ p = \text{Guest} \rightarrow \text{Date} \]

\[ q = \text{Date} \rightarrow \text{Room} \]

\[ s = \text{Guest} \rightarrow \text{Room} \]

<table>
<thead>
<tr>
<th>Guest</th>
<th>Date</th>
<th>Room</th>
<th>View</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>10 May</td>
<td>R1</td>
<td>Sea</td>
</tr>
<tr>
<td>G1</td>
<td>15 May</td>
<td>R2</td>
<td>Forest</td>
</tr>
<tr>
<td>G2</td>
<td>15 May</td>
<td>R1</td>
<td>Sea</td>
</tr>
<tr>
<td>G2</td>
<td>15 May</td>
<td>R1</td>
<td>Forest</td>
</tr>
</tbody>
</table>
Properly Embedded FD – Example

- The second relation can be obtained by projection from the first.
- We can discard the second, BUT:
  - We must keep the FD.
    - We can add it as a co-occurrence constraint.
    - The constraint, however, is not a key constraint and is “hard” to enforce.
  - There is another problem: redundancy – for each picky guest reservation we store the same room.

<table>
<thead>
<tr>
<th>Room</th>
<th>Date</th>
<th>Picky Guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>10 May</td>
<td>G1</td>
</tr>
<tr>
<td>R1</td>
<td>15 May</td>
<td>G1</td>
</tr>
<tr>
<td>R5</td>
<td>10 May</td>
<td>G3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Room</th>
<th>Picky Guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>G1</td>
</tr>
<tr>
<td>R5</td>
<td>G3</td>
</tr>
</tbody>
</table>

The transformation preserves information. The transformation preserves constraints.

Properly Embedded FD – Example
Properly Embedded FD – Definition

An FD $X \rightarrow Y$ (e.g., $C \rightarrow A$) is a properly embedded FD if:

- $R$ is the set of object sets of an edge (or eq. class) (e.g., $ABC$)
- $XY \subseteq R$ (e.g., $AC \subseteq ABC$)
- $X \cap Y = \emptyset$ (e.g., $A \cap C = \emptyset$)
- $X \rightarrow Y$ is an FD edge or implied by the FD edges (e.g., implied)
- Semantic equivalence holds for $X \rightarrow Y$ with respect to $R$
- $X^+ \not\subseteq R$ (e.g., $C^+ = ACD \not\subseteq ABC$)

Properly Embedded FD – Reduction

An edge $R$ (e.g., $ABC$) with a properly embedded FD $X \rightarrow Y$ (e.g., $C \rightarrow A$) is reduced as follows:

- Project $R$ on $R - Y$ (e.g., $ABC - A = BC$) and add it as a non-directed edge $E$.
- Discard $R$.
- Create a high-level relationship set $S$ over $E$ (e.g., $BC$) and the relevant edge set used to imply $X \rightarrow Y$ (e.g., $C \rightarrow D$ and $D \rightarrow A$) with $R$ (e.g., $ABC$) as its object sets.
- Specify the original FD of $R$ (e.g., $AB \rightarrow C$) as a co-occurrence constraint on $S$. 
Properly Embedded FDs in Equivalence Classes

- The FD $A \rightarrow B$ is embedded in the equivalence class \{AC, BC, D\}.
- We disconnect the head of the embedded FD, but keep its tail.
- Keeping the tail allows us to preserve information.
- We then make the transformed diagram preserve constraints.