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
  - Generation of basic constraints
  - design transformations
  - ORM hypergraph
  - Scheme generation
  - time/space adjustments
  - DB schemes and constraints
  - efficiently organized

ORM Hypergraphs

- A regular hypergraph with:
  - object sets as vertices
  - relationship sets as edges
  - Generalization/Specialization
  - Optional-participation markers
  - General constraints

Hypergraphs

- Graph = (V, E)
  - V = set of vertices
  - E = set of edges (two vertices; one for unordered loops)
  - e_1 : e_2 : ... : e_n
  - 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 = (x_1, ..., x_n)
  - e = E can be ordered: e = (x_1, ..., x_n, y_1, ..., y_k)
  - e = (x_1, ..., x_n, y_1, ..., y_k)

Conversion to ORM Hypergraph

ORM Hypergraphs

- A regular hypergraph with:
  - object sets as vertices
  - relationship sets as edges
- Plus:
  - Generalization/Specialization
  - Optional-participation markers
  - General constraints
- Example:

Functional Dependencies (FDs)

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

Let R be a relation scheme and let X \subseteq R and Y \subseteq R. X \rightarrow Y is a functional dependency or FD. A relation r(R) satisfies the FD X \rightarrow Y (or X \rightarrow Y 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) \cap \pi_X(r'), |\pi_Y(r) \cap \pi_Y(r')| = 1.
Chapter 9 - 11

### 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 → Y (sometimes written F → X → Y or F → X → Y) if X → Y holds for every relation that satisfies F.

#### Example of F

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

### 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 → Y; \} Let s[X] = t[X]. Then since Y → X, s[Y] = t[Y].

\{X → Y → Z; \} 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].

Thus, s[Y] = t[Y] and s[Z] = t[Z].

#### Proof of Completeness (Appendix C)

### Motivational Example

#### 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 → Y, we mean X → Y ⇒ F*.

#### Rules for computing F*:

1. (trivial implication) \( Y \subseteq X \to Y \)
2. (accumulation) \( X \to Y, W \to Z, W \subseteq Y \to X \to YZ \)
3. (projection) \( X \to Y, Z \subseteq Y \to X \to Z \)
4. \( \text{GuestNr} \cdot \text{Name City Room} \to \text{Name City Room} \)
5. \( \text{GuestNr} \cdot \text{Name City Room} \to \text{Name City Room} \)

We compute F* by a least-fixed-point process.
Additional FD-Implication Rules

(augmentation) \( X \rightarrow Y \rightarrow XZ \rightarrow YZ \)
Room \(\rightarrow\) Cost \(\rightarrow\) Room NrDays \(\rightarrow\) Cost NrDays
(transitivity) \( X \rightarrow Y \rightarrow Z \rightarrow X \rightarrow Z \)
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:
  \[ \begin{align*}
  R &= ABCD, F = \{ A \rightarrow B, B \rightarrow C \}, AD \rightarrow C \rightarrow F^*, BC \rightarrow A \rightarrow F^* \\
  1. A &\rightarrow B \quad \text{given} \\
  2. B &\rightarrow C \quad \text{given} \\
  3. A &\rightarrow C \quad \text{transitivity, 1 & 2} \\
  4. AD &\rightarrow CD \quad \text{augmentation, 3} \\
  5. AD &\rightarrow C \quad \text{projection, 4}
  \end{align*} \]

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:
  \[ \begin{align*}
  R &= ABCD, F = \{ A \rightarrow B, B \rightarrow C \}, AD \rightarrow C \rightarrow F^*, BC \rightarrow A \rightarrow F^* \\
  1. A &\rightarrow B \quad \text{given} \\
  2. B &\rightarrow C \quad \text{given} \\
  3. AD &\rightarrow AD \quad \text{T} \\
  4. AD &\rightarrow ABD \quad \text{A} \\
  5. AD &\rightarrow ABCD \quad \text{A} \\
  6. AD &\rightarrow C \quad \text{P}
  \end{align*} \]

- Examples:
  \[ \begin{align*}
  R &= ABCD, F = \{ A \rightarrow B, B \rightarrow C \} \\
  AD &= ABCD \\
  BC &= BC \\
  BD &= BCD \\
  D &= D \\
  A^* &= ABC
  \end{align*} \]

\( X \rightarrow Y \in F^* \) 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.
Reduction Transformations

- Discard "stuff" in model-theoretic view that we don't need
- Discard redundant relations (= reducible hypergraph edges)
- Discard implied constraints

Example

\[ \begin{array}{c}
\text{names}_\text{guest}_\text{occupying} = \text{names}_\text{guest}_\text{has}_{\text{Room}} \wedge \text{names}_\text{Name}\text{has}_{\text{Room}}
\end{array} \]

\[ \text{has}_{\text{Room}} \wedge \text{has}_{\text{Room}} \]

\[ \text{Occupies}_{\text{Room}} \]

\[ \text{Room}_{\text{Name}} \]

\[ \text{Name}_{\text{Room}} \]

\[ \text{Name}_{\text{Room}} \]

\[ \text{Name}_{\text{Room}} \]

Observe: 1. names_guest_occupying = names_guest_occupying \wedge \text{has}_{\text{Room}} \wedge \text{has}_{\text{Room}}

2. [Room \cdot \text{Guest}, \text{Guest} \cdot \text{Name}, \text{Name} \cdot \text{Room}] = \text{Room} \cdot \text{Name}

A transformation from M to M preserves constraints.

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 = \{\text{Room} \cdot \text{Guest}, \text{Guest} \cdot \text{Name}, \text{Name} \cdot \text{Room}, \text{Name} \cdot \text{Room}

\[ \forall x \forall y (\text{Name}(x) \land \text{names}_\text{guest}_\text{occupying}(\text{Room}(y)) \rightarrow \text{Name}(x) \land \text{Name}(y)) \]

Constraints of M', plus the constraints implied by P and P': C' = \{

\[ \text{Room} \cdot \text{Guest}, \text{Guest} \cdot \text{Name}, \text{Name} \cdot \text{Room}

\[ \forall x \forall y (\text{Name}(x) \land \text{names}_\text{guest}_\text{occupying}(\text{Room}(y)) \rightarrow \text{Name}(x) \land \text{Name}(y)) \]

\[ \forall y (\text{Guest}(x) \land \text{names}_\text{guest}_\text{occupying}(\text{Room}(y)) \rightarrow \text{Guest}(x) \land \text{Name}(y)) \]

\[ \forall y (\text{Guest}(x) \land \text{names}_\text{guest}_\text{occupying}(\text{Room}(y)) \rightarrow \text{Guest}(x) \land \text{Name}(y)) \]

\[ \forall y (\text{Guest}(x) \land \text{names}_\text{guest}_\text{occupying}(\text{Room}(y)) \rightarrow \text{Guest}(x) \land \text{Name}(y)) \]

\[ \forall y (\text{Guest}(x) \land \text{names}_\text{guest}_\text{occupying}(\text{Room}(y)) \rightarrow \text{Guest}(x) \land \text{Name}(y)) \]

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 = G, if F implies each FD in G and conversely.

\[ F = (A \cdot B, A_B, C, D) \land G = (A_B, C, D) \]

In F, A = ABCD \rightarrow A \cdot B & A_B \cdot D & C = CD \rightarrow C \cdot D.

In G, A_B = ABCD \rightarrow A \cdot B, AB^* = ABCD \rightarrow AB \cdot C, and

\[ F = G \]

\[ G = F \]

\[ F = G \]

\[ (G^* \cdot F \cdot F \cdot G^*) \rightarrow F = G^* \]

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:

\[ \text{Room} = \text{Room} \]

\[ \text{Guest} = \text{Guest} \]

\[ \text{Name} = \text{Name} \]

\[ \text{names}_\text{guest}_\text{occupying} = \text{names}_\text{guest}_\text{occupying} \wedge \text{has}_{\text{Room}} \wedge \text{has}_{\text{Room}} \]

Semantics

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}`

<table>
<thead>
<tr>
<th>Room Name</th>
<th>Guest</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>101</td>
<td>Smith</td>
</tr>
<tr>
<td>R3</td>
<td>103</td>
<td>Carter</td>
</tr>
</tbody>
</table>

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

If `r_1(R_1), ..., r_n(R_n)` and `u = r_1 | ... | r_n`, then `r_u = r_i`.

**Case 1.**

<table>
<thead>
<tr>
<th>Room</th>
<th>Guest</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>101</td>
<td>Smith</td>
</tr>
<tr>
<td>R3</td>
<td>103</td>
<td>Carter</td>
</tr>
</tbody>
</table>

**Case 2.**

<table>
<thead>
<tr>
<th>Room</th>
<th>Guest</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>101</td>
<td>Smith</td>
</tr>
<tr>
<td>R3</td>
<td>103</td>
<td>Carter</td>
</tr>
</tbody>
</table>

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

**Case 1:** `q` is name of guest staying in room.

Incongruent

**Case 2:** `s` = `q` = `Room` = `Name`

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.

Incongruent

**Case 2:** `s` = `q` = `Room` = `Name`

Observe that the FD `Room` - `Name` fails.

In general, we can ask, “Do we always get to the same place.”
Chapter 9 - 31

Congruency – Different Semantics

Case 2: q is name of room.

Incongruent

Congruent

Chapter 9 - 32

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

Chapter 9 - 33

Union-Covered Isolated Root Generalization

Partition-Covered Isolated Root Generalization

Chapter 9 - 34

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.

Chapter 9 - 35

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.
  - \( \pi_{\text{head}}(AB | I BC | J AD) \)
- Preserves constraints
  - Referential integrity and other constraints (trivially) OK

Chapter 9 - 36
**Head Reduction – Case 2 (Multiple Tails and Single Head)**

- Reduction:
  - Discard the head component.
  - Keep tails connected.
- Preserves information:
  - $\Delta \pi_{head}(AD \cup DC)$
  - Not enough (no B): $\pi_{head}(AD \cup DC)$
  - Not correct: $\pi_{head}(AD \cup DC \cup BE)$
- Preserves constraints: $A \Rightarrow D, D \Rightarrow C \Rightarrow AB \Rightarrow C$

**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.
  - $\Delta \pi_{head}(AF \cup FB \cup AC)$
- Preserves constraints:
  - $A \Leftrightarrow C \Rightarrow AB \Rightarrow C$
  - (note: forward transformation OK) $A \Rightarrow F, F \Rightarrow B, AB \Rightarrow C \Rightarrow A \Leftrightarrow C$
  - 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:
  - $\Delta \pi_{head}(AB \cup AC \cup AD \cup DC)$
  - Not enough (no B): $\pi_{head}(AB \cup AC \cup AD \cup DC)$
  - Not correct: $\pi_{head}(AB \cup AC \cup AD \cup DC \cup BE)$
- Preserves constraints:
  - $A \Rightarrow C \Rightarrow AB \Rightarrow C$
  - $A \Rightarrow D, D \Rightarrow C \Rightarrow A \Leftrightarrow C$

**FD Equivalence Classes**

- $X = Y$ for a set of FDs $F$ if $X \Rightarrow Y \in F^*$ and $Y \Rightarrow X \in F^*$.
  - $F = (A \Rightarrow BC, BC \Rightarrow D, D \Rightarrow A, A \Rightarrow D, BC \Rightarrow E)$
  - $A \Rightarrow D, A \Rightarrow BC, BC \Rightarrow D$, but not $BC \Rightarrow E$
  - $\Rightarrow$ 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

Composite Lexicalization

1-1 Nonlexical Object Sets

Further Head Reductions

Redundant NonFD Relationship Sets

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

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:

\[ AD = \frac{AD}{BC} \]

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

N-ary Relationship-Set Reduction

Non-reducible N-ary Edge

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.
  - 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.

Properly Embedded FD – Example

- The transformation preserves information.
- The transformation preserves constraints.
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 ⊆ 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 $R$
- $X' \subseteq R$ (e.g., $C' \subseteq ACD \subseteq ABC$)

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 $X \rightarrow Y$ with respect to $R$ (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.