Design Objectives

- Obtain the theoretically "best" design (normalize)
  - Remove redundancy and update anomalies
  - Remove nulls
  - Minimize the number of schemes and maximize their size
  - Make the design faithful to the specification
    - preserve information
    - preserve constraints
- Use cost analysis to adjust, if necessary (denormalize)
  - The theoretically "best" is often the best.
  - Adjust for application-dependent time and space considerations.

Update Anomalies

Modification Anomaly: e.g., change Kennedy to Clinton
- must update all redundant values consistently.
Deletion Anomaly: e.g., G4 cancels reservation – the fact that the Green room is room 5 is lost.
Insertion Anomaly: e.g., add a new room (the Gold room) – necessarily yields a null.

Update anomalies and redundancy are two sides of the same coin.

Join Dependencies – Definitions

- A join dependency (JD) denoted \( | \{R_1, \ldots, R_n\} \) holds for a relation \( r(R) \) if \( r = \pi_{R_1}^{-1}r \times \ldots \times \pi_{R_n}^{-1}r \).
  (e.g., \( |\{AB, AC\}\))

- When \( n = 2 \), we call a JD a Multivalued Dependency (MVD) and write \( X \rightarrow Y \) or \( X \rightarrow Z \) or \( X \rightarrow Y \mid Z \)
  where \( X = R_1 \cap R_2, Y = R_1 \setminus R_2 \), and \( Z = R_2 \setminus R_1 \).
  (e.g., \( A \rightarrow B \) or \( A \rightarrow C \) or \( A \rightarrow B \mid C \))

Redundancy

- We (usually) want to remove redundancy.
  - Space savings: no need to store duplicate values.
  - Time savings: no need for extra processing to avoid update anomalies.

  Basic Idea:
  - A data value \( v \) is redundant if we can "erase" \( v \) and then from the remaining data values and the constraints uniquely determine \( v \).
  - The constraints we consider: FDs, MVDs, JDs.

Join Dependencies – Example

Let \( r = A \times B \times C = A \times B \mid A \times C \)  
\begin{array}{ccc}
1 & a & x \\
1 & a & y \\
1 & b & x \\
1 & b & y \\
2 & a & y \\
2 & b & y \\
\end{array}

Note: \( r \) is the cross product of \( B \) and \( C \) w.r.t \( A \).

Observe: \( r = \pi_{AB}^{-1}r \times \pi_{AC}^{-1}r \)
This always holds when we build \( r \) by joining relationship sets in this way.

In general, however, if we arbitrarily create a relation, this may not happen. Add \( <2, a, x> \) to \( r \), for example, then \( r = \pi_{AB}^{-1}r \times \pi_{AC}^{-1}r \) because the join also yields \(<2, b, x>\), which is not in \( r \).

FD Redundancy

If \( B \rightarrow C \), the circled data values are redundant.

\begin{array}{ccc}
A & B & C \\
1 & 1 & 1 \\
2 & 1 & 1 \\
\end{array}
MVD Redundancy
If \( A \rightarrow B \mid C \), the circled data values are redundant.

\[
\begin{array}{ccc}
A & B & C \\
1 & 1 & 1 \\
1 & 2 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
A & B & C \\
1 & 2 & 1 \\
1 & 1 & 1 \\
1 & 2 & 2 \\
\end{array}
\]

JD Redundancy
If \( |\{AB, BC, AC\}| \), the circled data values are redundant.

\[
\begin{array}{ccc}
A & B & C \\
1 & 1 & 2 \\
2 & 1 & 1 \\
1 & 2 & 1 \\
\end{array}
\]

Minimize the Number of Schemes
- Combine object and relationship sets
- BUT only if there is no possibility of:
  - redundancy
  - nulls
- Preserve information and constraints

Sample Combinations
with Redundancy

Nulls

Incongruent

\[
\begin{array}{cccc}
A & B \\
1 & 1 \\
1 & 2 \\
3 & \downarrow \\
4 & \downarrow \\
5 & 1 \\
6 & \downarrow \\
\end{array}
\]

Congruent

\[
\begin{array}{cccc}
A & B \\
1 & 1 \\
2 & 1 \\
3 & 5 \\
4 & \downarrow \\
5 & \downarrow \\
6 & \downarrow \\
\end{array}
\]

Sample Combinations
with No Redundancy
Canonical ORM Hypergraph

- Congruent
- Nonrecursive
- Head and Tail Reduced
- Object-Set Reduced (Lexical & Merged)
- Non-FD-edge Reduced
- Embedded-FD Reduced
- Separately Linked (Semantically Separate Eq. Classes)
- Minimally Consolidated
- Semantically Head Consistent

Scheme Synthesis

- Input: a canonical ORM hypergraph.
- Output: a set of relation schemes with keys.

- Equivalence classes (including trivial equivalence classes) with FDs – each equivalence-class element is a key
- Nontrivial equivalence classes without FDs – each equivalence-class element is a key
- Non-FD edges – all the attributes together constitute a composite key
- Stand-alone object sets – the lone attribute is a key

Semantically Separate Eq. Class

HasName(Room Name) || WasNamed(Room Name) → Room Name

- R1: Kennedy → Nixon
- R2: Nixon → Kennedy
- R3: Carter → Carter
- R4: Blue → Green
- R5: Green → Blue

Room  Room Name  Prior Room Name
- R1: Kennedy  Nixon
- R2: Nixon  Kennedy
- R3: Carter  Carter
- R4: Blue  Green
- R5: Green  Blue

Scheme Synthesis – Example

- Case 1: A B C & D & E & F & D
- Case 2: B F G
- Case 3: C E
- Case 4: H

Inclusion Dependencies – Generation of Foreign Keys

- Input: a canonical ORM hypergraph and a set of schemes generated by the scheme-synthesis algorithm
- Output: a set of inclusion dependencies

- Generalization/specialization pairs
- Multiple appearances
- Subset constraints among relationship sets
Inclusion Dependencies – Example

\[ \forall x \forall y (D(x) \land E(y) \Rightarrow A(x) \land C(y)) \]

- Database scheme: \( q(A, B), r(A, C), s(D, E) \)
- Inclusion dependencies:
  - Case 1: \( qA \subseteq sD \land sA \subseteq sD \)
  - Case 2: \( qA = rA \)
  - Case 3: \( D(x) \subseteq sD \land E(y) \subseteq sE \)

B&B Example – Congruent

B&B Example – ORM Diagram

B&B Example – Canonical

B&B Example – ORM Hypergraph

B&B Example – Generated Database Scheme

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Room</td>
<td>(RoomNr, RoomName, NrBeds, Cost)</td>
</tr>
<tr>
<td>Guest</td>
<td>(GuestNr, GuestName, StreetNr, City)</td>
</tr>
<tr>
<td>Reservation</td>
<td>(GuestNr, RoomNr, ArrivalDate, NrDays)</td>
</tr>
</tbody>
</table>

\( q[RoomNr] \subseteq s[RoomNr] \Rightarrow s[Reservation[RoomNr]] \)

\( r[GuestNr] = s[Reservation[GuestNr]] \)
Keys and FDs

Let U be a set of object sets, and let F be a set of FDs over U. Let R \subseteq U be a relation scheme. A subset K of R (K need not be a proper subset of R) is a superkey of R if K is a candidate key of R if there does not exist a proper subset K' of K such that K' \subseteq R \subseteq F.

Example: U = ABCDE and F = \{A \rightarrow B, B \rightarrow A, AB \rightarrow C, D \rightarrow BC\}.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Candidate Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>A, B</td>
</tr>
<tr>
<td>CE</td>
<td>CE</td>
</tr>
<tr>
<td>ABCD</td>
<td>D</td>
</tr>
<tr>
<td>ABCDE</td>
<td>DE</td>
</tr>
</tbody>
</table>

Inextricably Embedded JDs

Generated Keys are Candidate Keys (Thm 10.3)

Superkeys:

- \{A, B\}
- \{A, C\}
- \{A, B\}
- \{A\}

Minimal Keys: Suppose not, then tail reducible.

No Nulls (Thm 10.5)

Canonical hypergraphs are congruent.

Generated Schemes have no Potential Redundancy* (Thm 10.4)

Canonical hypergraphs do not have edges that cause redundancy.

Synthesis Preserves Information (Thm 10.6)

Generated Scheme: A B C

Original Object and Relationship Sets:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Join/Project always returns the original.

*Except (possibly) for schemes that have a nontrivial, inextricably embedded JD.
Minimal Number of Schemes*  
(Thm 10.7)

A B C
1 1 0

A B C
1 1 1

BCA

*Without potential redundancy or nulls and assuming more than one tuple per relation is possible.

Cost Analysis  
Rule-of-Thumb Guidelines

- As a guide, consider denormalizing if:
  - nulls are applicable but unknown (e.g., address information)
  - redundancy is minimal and update anomalies are not expected (e.g., StreetNr City State - Zip)
  - replicated objects are large (e.g., images in View)
  - join frequencies are very high when compared to updates (e.g., approximate costs in foreign currencies)
- Using actual application characteristics, estimate space and time requirements for various possibilities and compare costs.

Minimal Number of Attributes in Schemes (Prop. 10.1 & 10.2)

- Hard to guarantee no fewer:
  - Do we count replacing two attributes, say Name and Address, by a single combined attribute, say Name-Address?
  - Perhaps a different way of deriving attributes for schemes might yield fewer.
- Can guarantee:
  - Proposition 10.1: We can’t make fewer by lexicalization or by one-to-one merges of nonlexical object sets.
  - Proposition 10.2: We can’t make fewer by consolidation within equivalence classes.

Cost Estimation – B & B (Space)

Assume: 5 rooms, 100 reservations, and 80 guests.

<table>
<thead>
<tr>
<th>Case</th>
<th>Room(RoomNr, RoomName, NrBeds, Cost)</th>
<th>Guest(GuestNr, GuestName, StreetNr, City)</th>
<th>Reservation(GuestNr, RoomNr, ArrivalDate, NrDays)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Room(RoomNr, RoomName, NrBeds, Cost)</td>
<td>Guest(GuestNr, GuestName, StreetNr, City)</td>
<td>Reservation(GuestNr, RoomNr, ArrivalDate, NrDays)</td>
</tr>
<tr>
<td>2</td>
<td>Room(RoomNr, RoomName, NrBeds, Cost)</td>
<td>Guest(GuestNr, GuestName, StreetNr, City)</td>
<td>Reservation(GuestNr, RoomNr, ArrivalDate, NrDays)</td>
</tr>
<tr>
<td></td>
<td>vs. Case 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vs. Case 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vs. Case 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cost Estimation – B & B (Time)

Assume indexed on primary keys.

1. (40%) What rooms are available?
2. (30%) Make a reservation.
3. (15%) Change a reservation.
4. (10%) Cancel a reservation.
5. (the rest) Miscellaneous.

Most important queries/updates:  
1. Case 1 & 2: Retrieve reservations that could overlap the requested date and determine room availability. (Case 1 insignificantly better.)  
2. Case 2 & 3: Retrieve and update guest data.

Cost Estimation – 

Synthesis Preserves Constraints  
(Thms 10.8 & 10.9)

- Theorem 10.8: We keep all constraints of the canonical hypergraph.
  - Some constraints become key constraints.
  - Some constraints become foreign-key constraints.
  - General constraints given or generated, including generated co-occurrence constraints for embedded FD reductions, remain intact.
- Theorem 10.9: Sometimes all constraints become key constraints or foreign-key constraints.
  - We can represent these constraints in SQL DDL.
  - Database systems efficiently check these constraints for us (no extra code need be written to check these constraints).
B & B – Semantic Change?
Case 1: Room(RoomNr, RoomName, NrBeds, Cost) vs. Case 2: Room(RoomNr, RoomName, NrBeds, Cost)
Reservation(GuestNr, RoomNr, ArrivalDate, NrDays)

Most important queries/updates:

1. (30%) Make a reservation.
2. Case 1: Insert tuple in Reservation (1 read & 1 write) and insert tuple in Guest, if necessary, (1 read and usually 1 write).
Case 2: Insert tuple in Reservation (1 read & 1 write); check duplicate guest information (read file, or add secondary index).

(Developer) Do we really need to check duplicate guest information? (Proprietor) Hmm, maybe not; it doesn't matter if it is different.
(Developer) Does a guest always need the same guest number? (Proprietor) Not really; there are no guest numbers in our manual system.
(Developer) Ahah! Great, this really lets us save – watch this.

Unique GuestNr in Reservation

Observe that we have a new equivalence class:

{ {GuestNr}, {RoomNr, ArrivalDate} }

And thus a new generated database scheme:
Reservation(GuestNr, GuestName, City, StreetNr, RoomNr, ArrivalDate, NrDays) Room(RoomNr, RoomName, NrBeds, Cost)

Redundancy in Nested Schemes

- The redundancy definition is the same as for flat relations.
- If a value change causes a constraint violation, the value is redundant.

 Nested Schemes

- Flat schemes often have replicated data values.
- Nested schemes allow us to collapse some of these replicated data values.

<table>
<thead>
<tr>
<th>NrBeds</th>
<th>RoomNr</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Nested Scheme Generation Example

1. NrBeds (RoomNr, RoomName, Cost (View)* (GuestNr, GuestName)*)
2. RoomNr, RoomName, Cost, NrBeds (View)* (GuestNr, GuestName)*
3. GuestNr GuestName RoomNr
   RoomNr, RoomName, Cost, NrBeds (View)*
Redundancy Prevention

This replication ... ... cause this redundancy.

Cost Analysis for Nested Schemes
- Nested schemes impose variable-length records.
- Recall variable-length record implementation strategies:
  - Reserve enough space for maximum.
  - Chain each nested record.
  - Reserve space for the expected number and chain the rest.
- Insertion, deletion, modification, retrieval tradeoffs.

Generalization of Algorithm 10.3 for N-ary Relationship Sets

- "Composite nodes" can be treated as a node (in Algorithm 10.3):
  - B C (A) (D)
  - D (B C); A B C
- NNF (see Exercise 10.35), basically:
  - Schemes should be constructed along hypergraph paths.
  - Schemes should not violate the natural 1-many hierarchical structure.

Guidelines for Selecting Nested Schemes
- Select "important nodes" as the initial nodes for nested-scheme generation – e.g., Scheme 3 or 2 in earlier Bed-&-Breakfast example.
- Maximize the size of schemes.
  - Select nodes included in the largest number of FD closures (i.e., when Algorithm 10.3 requires a new node to be arbitrarily selected, compute the set of unmarked nodes in the FD closure of every unmarked node and choose a node included in at least as many sets as any other node) – e.g., Scheme 1 in earlier example.
  - When possible, adjust these generated maximal schemes by placing the most important node first – e.g., Scheme 2 in earlier example.

Nested schemes impose variable-length records.
Recall variable-length record implementation strategies:
- Reserve enough space for maximum.
- Chain each nested record.
- Reserve space for the expected number and chain the rest.
- Insertion, deletion, modification, retrieval tradeoffs.

Composite nodes can be treated as a node (in Algorithm 10.3):
- B C (A) (D)
- D (B C); A B C
NNF (see Exercise 10.35), basically:
- Schemes should be constructed along hypergraph paths.
- Schemes should not violate the natural 1-many hierarchical structure.

Guidelines for Selecting Nested Schemes:
- Select "important nodes" as the initial nodes for nested-scheme generation – e.g., Scheme 3 or 2 in earlier Bed-&-Breakfast example.
- Maximize the size of schemes.
  - Select nodes included in the largest number of FD closures (i.e., when Algorithm 10.3 requires a new node to be arbitrarily selected, compute the set of unmarked nodes in the FD closure of every unmarked node and choose a node included in at least as many sets as any other node) – e.g., Scheme 1 in earlier example.
  - When possible, adjust these generated maximal schemes by placing the most important node first – e.g., Scheme 2 in earlier example.