Normal Forms

- Restrictions on the database schema
  - precludes certain undesirable properties from DB
  - NFs represent *good* DB design to be specified

- First normal form (1NF)
  - R is in 1NF if values in dom(A) are atomic for every attribute A in R: not multi-valued
  - accurate expressions of FDs
  - non-1NF relations force a complex DML to manipulate
  - is it reasonable to require 1NF?

```
document(title, author-list, date, keywords)
```

```
salesplan {Smith, Jones} 4/1/93 {profit, sales}
```

- transforming the document schema into 1NF can be done, but the results would not be user friendly
First Normal Form

- Normalization of document relation into 1NF

  document-1NF(title, author, day, month, year, keyword)

  (salesplan, Smith, 1, 4, 93, profit)
  (salesplan, Smith, 1, 4, 93, sales)
  (salesplan, Jones, 1, 4, 93, profit)
  (salesplan, Jones, 1, 4, 93, sales)

- Fourth normal form (4NF) will remove some of the awkwardness of such 1NF schema, but still complicates user interaction with the system

  title $\rightarrow\rightarrow$ author  
  title $\rightarrow\rightarrow$ keyword

- For new applications of relational databases that would benefit from non-1NF representation, new NFs and criteria for good design are necessary
Second Normal Form

- 2NF and 3NF arise from trying to avoid update anomalies and redundancy

<table>
<thead>
<tr>
<th>&lt;ex&gt;</th>
<th>Assign</th>
<th>(Flight</th>
<th>Day</th>
<th>Pilot</th>
<th>Gate)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>112</td>
<td>1 May</td>
<td>Jones</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>112</td>
<td>2 May</td>
<td>Cook</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>203</td>
<td>2 May</td>
<td>Jones</td>
<td>12</td>
</tr>
</tbody>
</table>

(Flight, Day) is a key for the relation

another FD: Flight → Gate

- update (Assign: 112, 3 May, Thomas, 8) violates the FD

- to avoid the violation, the system must scan the relation, and update the Gate# everywhere the Flight# appears

--- we don’t want to do it

- We want ...

  (1) change only one tuple

  (2) reduce (eliminate if possible) redundancy
Second Normal Form

In the example above, (Flight#, Gate#) is replicated in the relation, making the information redundant, and update complicated.

Solution: decomposition

<table>
<thead>
<tr>
<th>Passign</th>
<th>(Flight</th>
<th>Day</th>
<th>Pilot)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>112</td>
<td>1 May</td>
<td>Jones</td>
</tr>
<tr>
<td></td>
<td>112</td>
<td>2 May</td>
<td>Cook</td>
</tr>
<tr>
<td></td>
<td>203</td>
<td>2 May</td>
<td>Jones</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gassign</th>
<th>(Flight</th>
<th>Gate)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>112</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>203</td>
<td>12</td>
</tr>
</tbody>
</table>
Second Normal Forms

• Full dependency

  Given a set of FDs \( F \) and a FD \( X \rightarrow Y \) in \( F^+ \),
  \( Y \) is fully dependent on \( X \) if \( Y \) is not functionally
  dependent on any proper subset of \( X \).
  Otherwise, \( Y \) is partially dependent on \( X \).

<ex> FD={Flight Day \rightarrow Pilot Gate, Flight \rightarrow Gate}

  - Gate is partially dependent on (Flight Day)

  - Pilot is fully dependent on (Flight Day)

• Prime

  Attribute \( A \) is prime in relation schema \( R \) if \( A \) is a member of some candidate key of \( R \).
  Otherwise, \( A \) is non-prime in \( R \).
Second Normal Form

- **2NF**
  
  R is in 2NF w.r.t. a set of FDs F, if it is in 1NF and every non-prime attribute is fully dependent on every key of R

  <ex> Assign is not in 2NF,
  
  since Gate is partially dependent on (Flight Day).
  
  Passign and Gassign are in 2NF.

- **Requiring 2NF does not solve all the problems**

  Consider Student (S#, Name, Major-dept, Chair) with S# as key
  
  - no partial dependency, but remaining anomalies

  (1) To change Chair, all the tuples need to be changed

  (2) If a department has no student, can it exist?
  
  Can it have a chair?
Transitive Dependency and 3NF

- Why those problems are still there?

  - intuitively, the chair is an attribute of
    a department entity, not of a student entity
  
  - we have S# → Major-dept → Chair

- Transitive dependency

  An attribute A is transitively dependent on X
  if for some attribute Y, X → Y ∧ Y → A ∧ Y → X ∧ A ∉ XY

<ex> R = (Flight, Day, Pilot-id, Name)

FD = {FD → P, P → N, N → P}

Name is transitively dependent on (Flight, Day) because
FD → P, P → FD, P → N, N ∉ FDP

- 3NF

  R is in 3NF if no non-prime attribute of R is
  transitively dependent on any key of R.

  --- no transitive dependency
Third Normal Form

In the example above, R is not in 3NF because of a transitive dependency between Name and (Flight Day)

Solution: decomposition

\[ R_1 = (\text{Flight}, \text{Day}, \text{Pilot-id}) \]
\[ R_2 = (\text{Pilot-id}, \text{Name}) \]

Then \( R = \{R_1, R_2\} \) is in 3NF w.r.t. F

<Theorem> If R is in 3NF, then R is in 2NF.

Proof: If R is in 3NF but not in 2NF, then there must be a partial dependency \( X \rightarrow A \), where \( A \) is a non-prime and \( X \) is a proper subset of some key \( Y \).

Then \( Y \rightarrow X \) (\( Y \) is a key), \( X \rightarrow A \) (given), \( X \rightarrow Y \) (\( X \subset Y \)), and \( A \not\in X \) (\( A \) is non-prime)

\[ \Rightarrow A \text{ is transitively dependent on } Y \]

R is not in 3NF, a contradiction.
Notes on 3NF

- 3NF allows two FDs $X \rightarrow Y$ and $Y \rightarrow A$ in the same relation only if $X \rightarrow Y$ and $Y \rightarrow X$
  --- both $X$ and $Y$ are keys

- It leads to decomposing independent FDs into different relation schema, where they may be updated independently

- This suggests that more joins will be performed at run-time
  - system performance for such joins can be improved by physically storing such relations near to each other

- 1NF, 2NF, and 3NF are very popular
  - other normal forms and more theory exist, but only those simpler ones have achieved widespread acceptance
Good Database Design

- What is a good design?
  A 3NF decomposition that satisfies lossless join and dependency preservation
  --- certainly necessary, although not sufficient

- Constructing a 3NF database through decomposition
  - always possible to begin with any relation schema not in 3NF w.r.t. F, and decompose it into a schema that is in 3NF w.r.t. F
  - decomposing R means breaking it into $R_1$ and $R_2$ such that $\Pi_{R_1}(r) \times \Pi_{R_2}(r) = r$
    --- lossless join decomposition
  - decompose again repeatedly if either $R_1$ or $R_2$ is not in 3NF, until all relations are in 3NF w.r.t. F
Decomposition into 3NF

Given R and F

1. If there are no transitive dependencies in R, we are done

2. Otherwise, we have a transitive dependency on a key in R
   Let K be a key, Y ⊆ R, A a non-prime attribute in R, such that K→Y, Y→K, Y→A, and A ∉ KY

3. Let R₁=R-A and R₂=YA
   K is a key for R₁, Y is a key for R₂
   --- one transitive dependency is removed from R

4. Other attributes that are non-prime and ∈ R – (KY)
   that are dependent upon Y can be placed in R₂ at the same time
   --- speed up
Notes on Decomposition

- \( \Pi_{R_1}(r) \times \Pi_{R_2}(r) = r \) (lossless join)

Let \( R_1 \) and \( R_2 \) be relation schema with \( R_1 \cap R_2 = X \) where \( X \rightarrow R_1 \) (or \( R_2 \))
Then for any relation \( r \) with schema \( R_1R_2 \)
\[ r = \Pi_{R_1}(r) \times \Pi_{R_2}(r) \]

In step 3 of decomposition, \( R_1 = R - A, R_2 = YA \)
\( R_1 \cap R_2 = Y \)
\( Y \rightarrow A \) (given)
\( Y \rightarrow YA \) (augmentation)
Hence \( Y \rightarrow R_2 \)

- Decomposition cannot go on forever, since each time we decompose \( R \), resulting schema \( R_1 \) and \( R_2 \) are smaller, and no transitive dependency can exist with only two attributes

- Decomposition is not unique