A “Reducible” Relation

WIFE_OF_HENRY_VIII

<table>
<thead>
<tr>
<th>Wife#</th>
<th>FirstName</th>
<th>LastName</th>
<th>Fate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Catherine</td>
<td>of Aragon</td>
<td>divorced</td>
</tr>
<tr>
<td>2</td>
<td>Anne</td>
<td>Boleyn</td>
<td>beheaded</td>
</tr>
<tr>
<td>3</td>
<td>Jane</td>
<td>Seymour</td>
<td>died</td>
</tr>
<tr>
<td>4</td>
<td>Anne</td>
<td>of Cleves</td>
<td>divorced</td>
</tr>
<tr>
<td>5</td>
<td>Catherine</td>
<td>Howard</td>
<td>beheaded</td>
</tr>
<tr>
<td>6</td>
<td>Catherine</td>
<td>Parr</td>
<td>survived</td>
</tr>
</tbody>
</table>

(Note the underscoring of the primary key attribute.)

“Decomposing” H8’s Wives

<table>
<thead>
<tr>
<th>Wife#</th>
<th>FirstName</th>
<th>LastName</th>
<th>Fate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Catherine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Anne</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Jane</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Anne</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Catherine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Catherine</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A Join Dependency That Does Not Hold

Although W_FN is irreducible, we can of course take several projections of it, the following two in particular:

W_FN { Wife# }       W_FN { FirstName }

Wife# | FirstName | Wife# | LastName | Fate  |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Catherine</td>
<td>1</td>
<td>of Aragon</td>
<td>divorced</td>
</tr>
<tr>
<td>2</td>
<td>Anne</td>
<td>2</td>
<td>Boleyn</td>
<td>beheaded</td>
</tr>
<tr>
<td>3</td>
<td>Jane</td>
<td>3</td>
<td>Seymour</td>
<td>died</td>
</tr>
<tr>
<td>4</td>
<td>Anne</td>
<td>4</td>
<td>of Cleves</td>
<td>divorced</td>
</tr>
<tr>
<td>5</td>
<td>Catherine</td>
<td>5</td>
<td>Howard</td>
<td>beheaded</td>
</tr>
<tr>
<td>6</td>
<td>Catherine</td>
<td>6</td>
<td>Parr</td>
<td>survived</td>
</tr>
</tbody>
</table>

But the JOIN of these two does not yield W_FN, so the JD *{ { Wife# }, { FirstName} } does not hold in W_FN.

Decomposition of W_LN_F

W_LN_F can be further decomposed:

W_LN | W_F

<table>
<thead>
<tr>
<th>Wife#</th>
<th>LastName</th>
<th>Wife#</th>
<th>Fate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>of Aragon</td>
<td>1</td>
<td>divorced</td>
</tr>
<tr>
<td>2</td>
<td>Boleyn</td>
<td>2</td>
<td>beheaded</td>
</tr>
<tr>
<td>3</td>
<td>Seymour</td>
<td>3</td>
<td>died</td>
</tr>
<tr>
<td>4</td>
<td>of Cleves</td>
<td>4</td>
<td>divorced</td>
</tr>
<tr>
<td>5</td>
<td>Howard</td>
<td>5</td>
<td>beheaded</td>
</tr>
<tr>
<td>6</td>
<td>Parr</td>
<td>6</td>
<td>survived</td>
</tr>
</tbody>
</table>
3-Way Join Dependency

So the following JD holds in W_LN_F:

\* \{ \text{Wife#}, \text{LastName} \}; \{ \text{Wife#}, \text{Fate} \} \]

and we can conclude that the following 3-way JD holds in WIFE_OF_HENRY_VIII:

\* \{ \text{Wife#}, \text{FirstName} \}; \{ \text{Wife#}, \text{LastName} \}; \{ \text{Wife#}, \text{Fate} \} \]

i.e., WIFE_OF_HENRY_VIII =

WIFE_OF_HENRY_VIII \{ \text{Wife#}, \text{FirstName} \} \text{JOIN} \\
WIFE_OF_HENRY_VIII \{ \text{Wife#}, \text{LastName} \} \text{JOIN} \\
WIFE_OF_HENRY_VIII \{ \text{Wife#}, \text{Fate} \} 

Conclusion

In the example at hand, the single relvar design is preferred, so long as the constraint implied by it truly reflects the real world. (For example, if it turns out that in fact not every wife has a last name, then we should separate out W_LN.)

But the example at hand is rather special:
- Each operand of the 3-way JD that holds in WIFE_OF_HENRY_VIII includes a key of that relvar
- and it is the same key in each case, viz. \{\text{Wife#}\}

We need to look at some not-so-special examples.

Design Comparison

Does the decomposed design have any advantages?

The single relvar design carries an implicit constraint to the effect that every wife has a wife number, a first name, a last name and a fate.

This constraint is not implicit in the decomposed design.

In fact, to enforce it, each of W_FN, W_LN and W_F needs a foreign key referencing each of the other two.

But then the first attempt to insert a tuple into any of them must fail (unless multiple assignment is available).

A Not-So-Special JD

Recall that we decided to “split” (i.e., decompose) this one, as follows …

Functional Dependency

Because \{\text{StudentId}\} is a key of the projection ENROLMENT [\text{StudentId}, \text{Name}], we say that the following functional dependency (FD) holds in ENROLMENT:

\{ \text{StudentId} \} \rightarrow \{ \text{Name} \}

- The arrow, pronounced “determines”, indicates an FD.
- Each operand is a set of attributes (hence the braces).

Name is a function of StudentId.
For each StudentId there is exactly one Name.

Splitting ENROLMENT

Notice the JD: \* \{ \text{StudentId, Name}, \text{Courseld} \}

that holds in ENROLMENT.
Anatomy of an FD

A → B

Reminder: The determinant is a set of attributes, and so is the dependant set.

P.S. “dependant” is not a misspelling! It’s the noun, not the adjective.

FDs That Do Not Hold in ENROLMENT

{ Name } → { StudentId }
{ Name } → { CourseId }
{ CourseId } → { StudentId }
{ StudentId, Name } → { CourseId }
{ StudentId } → { CourseId }
{ StudentId, Name } → { CourseId }

also: { x } → { StudentId }, because x is not an attribute of ENROLMENT.

Theorems About FDs

Assume A → B holds in r. Then:

Left-Augmentation: If A' is a superset of A, then A' → B holds in r.
Right-reduction: If B' is a subset of B, then A → B' holds in r.
Transitivity: If A → B and B → C hold in r, then A → C holds in r.
In general: If A → B and C → D hold in r, then:
A \cup (C - B) → B \cup D holds in r.

Left-Irreducibility

If A → B holds in r and there is no proper subset A' of A such that
A' → B holds in r, then A → B is a left-irreducible FD (in r).
In this case, B is sometimes said to be fully dependent on A.

Conversely, if there is such a subset A', then A → B is a left-
irreducible FD (in r), and B is therefore not fully dependent on A.

FDs and Keys

If A → B is an FD in r and A \cup B constitutes the entire heading of
r, then A is a superkey of r.

If A → B is a left-irreducible FD in r and A \cup B constitutes the
entire heading of r, then A is a key of r.

(The longer term candidate key is often used instead of key, for
historical reasons.)

Normal Forms

Arising from the study of JDs in general and FDs in particular,
various “normal forms” have been defined:
• First Normal Form (1NF)
• Second Normal Form (2NF)
• Third Normal Form (3NF)
• Boyce/Codd Normal Form (BCNF)
• Fourth Normal Form (4NF)
• Fifth Normal Form (5NF)
• Sixth Normal Form (6NF)

Each of these is in a sense stricter than its immediate predecessor.
The ones shown in bold are particularly important. The others
were early attempts that eventually proved inadequate.
Normalisation is the act of decomposing a relvar that fails to satisfy a given normal form (e.g., BCNF) such that the result is an equivalent set of two or more “smaller” relvars that do satisfy that normal form.

We decompose by taking the projections specified in a given join dependency (JD).

In the case of ENROLMENT, the given JD is

\[*\{ [StudentId, Name], [StudentId, CourseId] \}*

determined by the FD \{ StudentId \} \rightarrow \{ Name \}

Purposes of Normalisation

A database all of whose relvars satisfy 6NF has the following possibly desirable properties:

- No redundancy (i.e., no recording of the same information more than once)
- No “update anomalies” (to be explained later)
- Orthogonality (independent recording of the simplest facts)

But 5NF is usually sufficient (and 6NF is sometimes problematical with existing technology), as we shall see …