Normal Forms

- Each normal form is a set of conditions on a schema that guarantees certain properties (relating to redundancy and update anomalies)
- First normal form (1NF) is the same as the definition of relational model (relations = sets of tuples; each tuple = sequence of atomic values)
- Second normal form (2NF) – no partial dependency
- The two commonly used normal forms are third normal form (3NF) and Boyce-Codd normal form (BCNF)
- Normalization is a database design technique for producing a set of suitable relations that support the data requirements of an enterprise.
How Normalization Supports Database Design

Relationship Between Normal Forms
Process of Normalization

Un-Normalized Form (UNF)

- A table that contains one or more repeating groups.
- To create an unnormalized table:
  - Transform data from information source (e.g. form) into table format with columns and rows.

<table>
<thead>
<tr>
<th>propertyNo</th>
<th>pAddress</th>
<th>iDate</th>
<th>iTTime</th>
<th>comments</th>
<th>staffNo</th>
<th>sName</th>
<th>carReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG4</td>
<td>6 Lawrence St, Glasgow</td>
<td>18-Oct-00</td>
<td>10.00</td>
<td>Need to replace crockery</td>
<td>SG37</td>
<td>Ann Beech</td>
<td>M231 JGR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22-Apr-01</td>
<td>09.00</td>
<td>In good order</td>
<td>SG14</td>
<td>David Ford</td>
<td>M533 HDR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1-Oct-01</td>
<td>12.00</td>
<td>Damp rot in bathroom</td>
<td>SG14</td>
<td>David Ford</td>
<td>N721 HFR</td>
</tr>
<tr>
<td>PG16</td>
<td>5 Novar Dr, Glasgow</td>
<td>22-Apr-01</td>
<td>13.00</td>
<td>Replace living room carpet</td>
<td>SG14</td>
<td>David Ford</td>
<td>M533 HDR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24-Oct-01</td>
<td>14.00</td>
<td>Good condition</td>
<td>SG37</td>
<td>Ann Beech</td>
<td>N721 HFR</td>
</tr>
</tbody>
</table>
First Normal Form (1NF)

- A relation in which intersection of each row and column contains one and only one (atomic) value.

<table>
<thead>
<tr>
<th>propertyNo</th>
<th>lDate</th>
<th>lTime</th>
<th>pAddress</th>
<th>comments</th>
<th>staffNo</th>
<th>sName</th>
<th>carReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG4</td>
<td>1-Oct-00</td>
<td>10:00</td>
<td>6 Lawrence St,</td>
<td>Need to replace crockery</td>
<td>SG37</td>
<td>Ann Beech</td>
<td>M231 JGR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Glasgow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PG4</td>
<td>22-Apr-01</td>
<td>09:00</td>
<td>6 Lawrence St,</td>
<td>In good order</td>
<td>SG14</td>
<td>David Ford</td>
<td>M333 HDR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Glasgow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PG4</td>
<td>1-Oct-01</td>
<td>12:00</td>
<td>6 Lawrence St,</td>
<td>Dump not in bathroom</td>
<td>SG14</td>
<td>David Ford</td>
<td>N721 HFR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Glasgow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PG6</td>
<td>22-Apr-01</td>
<td>13:00</td>
<td>5 Novar Dr,</td>
<td>Replace living room</td>
<td>SG14</td>
<td>David Ford</td>
<td>M333 HDR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Glasgow</td>
<td>carpet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PG6</td>
<td>24-Oct-01</td>
<td>14:00</td>
<td>5 Novar Dr,</td>
<td>Good condition</td>
<td>SG37</td>
<td>Ann Beech</td>
<td>N721 HFR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Glasgow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Second Normal Form (2NF)

- Based on concept of full functional dependency:
  - A, X and B are attributes of a relation,
  - B is fully dependent on (A, X) if B is functionally dependent on (A,X) but not on any proper subset of (A,X) such as (A) or (X).
  - A, X → B and there is NO A → B or X → B

- 2nd Normal Form
  - A relation that does not have a FD, X → Y, where X is a strict subset of that schema’s key and Y has attributes that do not occur in any of the schema’s keys.
1NF to 2NF (Functional Dependencies)

- Fd1: PropertyNo, iDate → iTime, staffNo, comments, sName, carReg
- Fd2: PropertyNo → pAddress
- Fd3: staffNo → sName
- Fd4: iDate, staffNo → carReg
- Fd5: iDate, iTime, carReg → all other attributes
- Fd6: iDate, iTime, staffNo → all other attributes

Transformed into following two tables.

- Property (propertyNo, pAddress)
- PropertyInspection (propertyNo, iDate, iTime, comments, staffNo, sName, carReg)
**BCNF**

- **Definition:** A relation schema $R$ is in BCNF if for every FD $X \rightarrow Y$ associated with $R$ either
  - $Y \subseteq X$ (i.e., the FD is trivial) or
  - $X$ is a superkey of $R$

- **Example:** Person1 ($SSN$, $Name$, $Address$)
  - The only FD is $SSN \rightarrow Name$, $Address$
  - Since $SSN$ is a key, Person1 is in BCNF

**(non) BCNF Examples**

- **Person ($SSN$, $Name$, $Address$, $Hobby$)**
  - The FD $SSN \rightarrow Name$, $Address$ does not satisfy requirements of BCNF
  - Since the key is ($SSN$, $Hobby$)

- **HasAccount ($AcctNum$, $ClientId$, $Officeld$)**
  - The FD $AcctNum \rightarrow Officeld$ does not satisfy BCNF requirements
  - Since keys are ($ClientId$, $Officeld$) and ($AcctNum$, $ClientId$); not $AcctNum$. 
Redundancy

- Suppose R has a FD $A \rightarrow B$, and $A$ is not a superkey. If an instance has 2 rows with same value in $A$, they must also have same value in $B$ (=> redundancy, if the $A$-value repeats twice)

- If $A$ is a superkey, there cannot be two rows with same value of $A$
  - Hence, BCNF eliminates redundancy

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td>stamps</td>
</tr>
<tr>
<td>1111</td>
<td>Joe</td>
<td>123 Main</td>
<td>coins</td>
</tr>
</tbody>
</table>

Third Normal Form

- A relational schema R is in 3NF if for every FD $X \rightarrow Y$ associated with R either:
  - Every $A \in Y$ is part of some key of R
  - 3NF is weaker than BCNF (every schema that is in BCNF is also in 3NF)
3NF Example

- HasAccount (AcctNum, ClientId, OfficeId)
  - ClientId, OfficeId → AcctNum
    - OK since LHS contains a key
  - AcctNum → OfficeId
    - OK since RHS is part of a key

- HasAccount is in 3NF but it might still contain redundant information due to AcctNum → OfficeId (which is not allowed by BCNF)

3NF (Non) Example

- Person (SSN, Name, Address, Hobby)
  - (SSN, Hobby) is the only key.
  - SSN → Name violates 3NF conditions since Name is not part of a key and SSN is not a superkey
Decompositions

- **Goal**: Eliminate redundancy by decomposing a relation into several relations in a higher normal form

- Decomposition must be *lossless*: it must be possible to reconstruct the original relation from the relations in the decomposition

Decomposition

- **Schema** \( R = (R, F) \)
  - \( R \) is a set of attributes
  - \( F \) is a set of functional dependencies over \( R \)
    - Each key is described by a FD

- The *decomposition of schema* \( R \) is a collection of schemas \( R_i = (R_i, F_i) \) where
  - \( R = \bigcup_i R_i \) for all \( i \) *(no new attributes)*
  - \( F_i \) is a set of functional dependences involving only attributes of \( R_i \)
  - \( F \) entails \( F_i \) for all \( i \) *(no new FDs)*

- The *decomposition of an instance*, \( r \), of \( R \) is a set of relations \( r_i = \pi_{R_i}(r) \) for all \( i \)
Example Decomposition

Schema \((R, F)\) where
\[ R = \{\text{SSN, Name, Address, Hobby}\} \]
\[ F = \{\text{SSN} \rightarrow \text{Name, Address}\} \]
can be decomposed into:
\[ R_1 = \{\text{SSN, Name, Address}\} \]
\[ F_1 = \{\text{SSN} \rightarrow \text{Name, Address}\} \]
and
\[ R_2 = \{\text{SSN, Hobby}\} \]
\[ F_2 = \{\} \]

Lossless Schema Decomposition

- A decomposition should not lose information
- A decomposition \((R_1, \ldots, R_n)\) of a schema, \(R\), is lossless if every valid instance, \(r\), of \(R\) can be reconstructed from its components:
\[ r = r_1 \Join r_2 \Join \ldots \Join r_n \]
where each \(r_i = \pi_{R_i}(r)\)
Lossy Decomposition

- The following is always the case (Think why?):
  \[ r \subseteq r_1 \quad \cap \quad r_2 \quad \cap \quad \ldots \quad \cap \quad r_n \]

- But the following is not always true:
  \[ r \supseteq r_1 \quad \cap \quad r_2 \quad \cap \quad \ldots \quad \cap \quad r_n \]

- Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Joe</td>
<td>1 Pine</td>
</tr>
<tr>
<td>2222</td>
<td>Alice</td>
<td>2 Oak</td>
</tr>
<tr>
<td>3333</td>
<td>Alice</td>
<td>3 Pine</td>
</tr>
</tbody>
</table>

The tuples \((2222, Alice, 3 Pine)\) and \((3333, Alice, 2 Oak)\) are in the join, but not in the original.

Lossy Decompositions: What is Actually Lost?

- In the previous example, the tuples \((2222, Alice, 3 Pine)\) and \((3333, Alice, 2 Oak)\) were gained, not lost!
  - Why do we say that the decomposition was lossy?

- What was lost is information:
  - That 2222 lives at 2 Oak: In the decomposition, 2222 can live at either 2 Oak or 3 Pine
  - That 3333 lives at 3 Pine: In the decomposition, 3333 can live at either 2 Oak or 3 Pine
Testing for Losslessness

A (binary) decomposition of $R = (R, F)$ into $R_1 = (R_1, F_1)$ and $R_2 = (R_2, F_2)$ is lossless if and only if:

- either the FD $(R_1 \cap R_2) \rightarrow R_1$ is in $F^+$
- or the FD $(R_1 \cap R_2) \rightarrow R_2$ is in $F^+$

Example

Schema $(R, F)$ where

$R = \{\text{SSN, Name, Address, Hobby}\}$

$F = \{\text{SSN} \rightarrow \text{Name, Address}\}$

can be decomposed into:

$R_1 = \{\text{SSN, Name, Address}\}$

$F_1 = \{\text{SSN} \rightarrow \text{Name, Address}\}$

and

$R_2 = \{\text{SSN, Hobby}\}$

$F_2 = \{\}$

Since $R_1 \cap R_2 = \text{SSN}$ and $\text{SSN} \rightarrow R_1$ the decomposition is lossless
Intuition Behind the Test for Losslessness

- Suppose $R_1 \cap R_2 \rightarrow R_2$. Then a row of $r_1$ can combine with exactly one row of $r_2$ in the natural join (since in $r_2$ a particular set of values for the attributes in $R_1 \cap R_2$ defines a unique row).

Tuple Structure in a Lossless Binary Decomposition
Proof of Lossless Condition

- \( r \subseteq r_1 \bowtie r_2 \) — this is true for any decomposition
- \( r \supseteq r_1 \bowtie r_2 \)

If \( R_1 \cap R_2 \rightarrow R_2 \) then
\[
\text{card} (r_1 \bowtie r_2) = \text{card} (r_1)
\]
(since each row of \( r_1 \) joins with exactly one row of \( r_2 \))

But \( \text{card} (r) \geq \text{card} (r_1) \) (since \( r_1 \) is a projection of \( r \))
and therefore \( \text{card} (r) \geq \text{card} (r_1 \bowtie r_2) \)

Hence \( r = r_1 \bowtie r_2 \)

Dependency Preservation

- Consider a decomposition of \( R = (R, F) \) into \( R_1 = (R_1', F_1) \) and \( R_2 = (R_2', F_2) \)
  - An FD \( X \rightarrow Y \) of \( F^* \) is in \( F_1 \) iff \( X \cup Y \subseteq R_i \)
  - An FD, \( f \in F^* \) may be in neither \( F_1 \), nor \( F_2 \), nor even \( (F_1 \cup F_2)^+ \)
    - Checking that \( f \) is true in \( r_1 \) or \( r_2 \) is (relatively) easy
    - Checking \( f \) in \( r_1 \cup r_2 \) is harder — requires a join
    - Ideally: want to check FDs locally, in \( r_1 \) and \( r_2 \), and have a guarantee that
      every \( f \in F \) holds in \( r_1 \bowtie r_2 \)
- The decomposition is dependency preserving iff the sets \( F \) and \( F_1 \cup F_2 \) are equivalent: \( F^* = (F_1 \cup F_2)^+ \)
  - Then checking all FDs in \( F \) as \( r_1 \) and \( r_2 \) are updated, can be done by checking \( F_1 \) in \( r_1 \) and \( F_2 \) in \( r_2 \)
Dependency Preservation

- If \( f \) is an FD in \( F \), but \( f \) is not in \( F_1 \cup F_2 \), there are two possibilities:
  - \( f \in (F_1 \cup F_2)^+ \)
    - If the constraints in \( F_1 \) and \( F_2 \) are maintained, \( f \) will be maintained automatically.
  - \( f \notin (F_1 \cup F_2)^+ \)
    - \( f \) can be checked only by first taking the join of \( r_1 \) and \( r_2 \). This is costly.
    - Incur additional runtime overhead of constraint maintenance.

Example

Schema \((R, F)\) where
\[
\begin{align*}
R &= \{ \text{SSN, Name, Address, Hobby} \} \\
F &= \{ \text{SSN} \rightarrow \text{Name, Address} \}
\end{align*}
\]
can be decomposed into:
\[
\begin{align*}
R_1 &= \{ \text{SSN, Name, Address} \} \\
F_1 &= \{ \text{SSN} \rightarrow \text{Name, Address} \}
\end{align*}
\]
and
\[
\begin{align*}
R_2 &= \{ \text{SSN, Hobby} \} \\
F_2 &= \{ \}
\end{align*}
\]
Since \( F = F_1 \cup F_2 \) the decomposition is dependency preserving.
Example

- Schema: \((ABC; F), F = \{A \rightarrow B, B \rightarrow C, C \rightarrow B\}\)

- Decomposition:
  - \((AC, F_1), F_1 = \{A \rightarrow C\}\)
    - Note: \(A \rightarrow C \notin F\), but in \(F^+\)
  - \((BC, F_2), F_2 = \{B \rightarrow C, C \rightarrow B\}\)

- \(A \rightarrow B \notin (F_1 \cup F_2), \text{ but } A \rightarrow B \in (F_1 \cup F_2)^+\).
  - So \(F^+ = (F_1 \cup F_2)^+\) and thus the decompositions is still dependency preserving

Example

- HasAccount \((AcctNum, ClientId, OfficeId)\)
  - \(f_1: AcctNum \rightarrow OfficeId\)
  - \(f_2: ClientId, OfficeId \rightarrow AcctNum\)

- Decomposition:
  - \(R_1 = (AcctNum, OfficeId); \{AcctNum \rightarrow OfficeId\}\)
  - \(R_2 = (AcctNum, ClientId; \{}\)\)

- Decomposition is lossless:
  - \(R_1 \cap R_2 = \{AcctNum\}\) and \(AcctNum \rightarrow OfficeId\)

- In BCNF

- Not dependency preserving: \(f_2 \notin (F_1 \cup F_2)^+\)

- HasAccount does not have BCNF decompositions that are both lossless and dependency preserving! (check by enumeration)

- Hence: “BCNF + lossless + dependency preserving” decompositions are not always achievable!
BCNF Decomposition Algorithm

**Input:** \( R = (R; F) \)

\( \text{Decomp} := R \)

**while** there is \( S = (S; F') \in \text{Decomp} \) and \( S \) not in BCNF **do**

Find \( X \to Y \in F' \) that violates BCNF \// i.e., \( X \) isn’t a superkey in \( S \)

Replace \( S \) in \( \text{Decomp} \) with \( S_1 = (XY; F_1), \ S_2 = (S - (Y - X); F_2) \)

\// \( F_1 = \) all FDs of \( F' \) involving only attributes of \( XY \)

\// \( F_2 = \) all FDs of \( F' \) involving only attributes of \( S - (Y - X) \)

**end**

return \( \text{Decomp} \)

---

Simple Example

- **HasAccount:**

  \((\text{ClientId}, \text{Officeld, AcctNum})\)  
  \(\text{ClientId,Officeld} \rightarrow \text{AcctNum}\)  
  \(\text{AcctNum} \rightarrow \text{Officeld}\)

- **Decompose using **\(\text{AcctNum} \rightarrow \text{Officeld} :\)**

  \((\text{Officeld, AcctNum})\)  
  \(\text{BCNF: AcctNum is key}\)  
  \(\text{FD: AcctNum} \rightarrow \text{Officeld}\)  
  \(\text{BCNF (only trivial FDs)}\)
A Larger Example

Given: \( R = (R; F) \) where \( R = ABCDEGHK \) and
\[
F = \{ABH \rightarrow C, A \rightarrow DE, BGH \rightarrow K, K \rightarrow ADH, BH \rightarrow GE\}
\]

Step 1: Find a FD that violates BCNF
- Not \( ABH \rightarrow C \) since \( (ABH)^+ \) includes all attributes
  - (BH is a key)
- \( A \rightarrow DE \) violates BCNF since \( A \) is not a superkey \( (A^+ = ADE) \)

Step 2: Split \( R \) into:
- \( R_1 = (ADE, F_1 = \{A \rightarrow DE\}) \)
- \( R_2 = (ABCGHK; F_2 = \{ABH \rightarrow C, BGH \rightarrow K, K \rightarrow AH, BH \rightarrow G\}) \)
  - Note 1: \( R_1 \) is in BCNF
  - Note 2: Decomposition is lossless since \( A \) is a key of \( R_1 \)
  - Note 3: FDs \( K \rightarrow D \) and \( BH \rightarrow E \) are not in \( F_1 \) or \( F_2 \). But both can be derived from \( F_1 \cup F_2 \)
    - (E.g., \( K \rightarrow A \) and \( A \rightarrow D \) implies \( K \rightarrow D \))
  - Hence, decomposition is dependency preserving.

Example (con’t)

Given: \( R_2 = (ABCGHK; \{ABH \rightarrow C, BGH \rightarrow K, K \rightarrow AH, BH \rightarrow G\}) \)

step 1: Find a FD that violates BCNF.
- Not \( ABH \rightarrow C \) or \( BGH \rightarrow K \), since \( BH \) is a key of \( R_2 \)
- \( K \rightarrow AH \) violates BCNF since \( K \) is not a superkey \( (K^+ = AHK) \)

step 2: Split \( R_2 \) into:
- \( R_{21} = (KAH, F_{21} = \{K \rightarrow AH\}) \)
- \( R_{22} = (BCGK; F_{22} = \{\}) \)

  - Note 1: Both \( R_{21} \) and \( R_{22} \) are in BCNF.
  - Note 2: The decomposition is lossless (since \( K \) is a key of \( R_{21} \))
  - Note 3: FDs \( ABH \rightarrow C, BGH \rightarrow K, BH \rightarrow G \) are not in \( F_{21} \) or \( F_{22} \), and they can’t be derived from \( F_1 \cup F_{21} \cup F_{22} \).
  - Hence the decomposition is not dependency-preserving.
Properties of BCNF Decomposition Algorithm

Let $X \rightarrow Y$ violate BCNF in $R = (R,F)$ and $R_1 = (R_1,F_1)$,
$R_2 = (R_2,F_2)$ is the resulting decomposition. Then:

- There are fewer violations of BCNF in $R_1$ and $R_2$ than there were in $R$
  - $X \rightarrow Y$ implies $X$ is a key of $R_1$
  - Hence $X \rightarrow Y \in F_1$ does not violate BCNF in $R_1$ and, since $X \rightarrow Y \notin F_2$, does not violate BCNF in $R_2$ either
  - Suppose $f : X' \rightarrow Y' \in F$ doesn’t violate BCNF in $R$. If $f \notin F_1$ or $F_2$ it does not violate BCNF in $R_1$ or $R_2$ either since $X'$ is a superkey of $R$ and hence also of $R_1$ and $R_2$.

Properties of BCNF Decomposition Algorithm

- A BCNF decomposition is not necessarily dependency preserving
- But always lossless:
  - since $R_1 \cap R_2 = X$, $X \rightarrow Y$, and $R_2 = XY$
- BCNF+lossless+dependency preserving is sometimes unachievable (recall HasAccount)