Big Idea “Measure Twice, Cut Once”

E/R is mostly a visualization technique

Poor schemas can lead to inconsistency and performance inefficiencies

Updating a schema is expensive

Identify functional dependencies and normalize to **make well-behaved and fast databases the first time**
Motivating Example

We want to store information about **people** (Name, SSN, PhoneNumber, City)

Known properties:
- Each person may have multiple phones
- Each person lives in only one city
Motivating Example

Is this a good representation of **people**?

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-123-4567</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-890-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>626-246-8024</td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>
Motivating Example

Why is this a poor representation of people?

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-123-4567</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-890-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>626-246-8024</td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>

Anomalies:
- **Redundancy** (data for Fred is duplicated)
- **Slow Updates** (what if Fred moved to Oahu?)
- **Zealous Deletion** (what if Joe got rid of his phone?)
Motivating Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-123-4567</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-890-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>626-246-8024</td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>

 normalization!
## Motivating Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>PhoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>206-123-4567</td>
</tr>
<tr>
<td>123-45-6789</td>
<td>206-890-1234</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>626-246-8024</td>
</tr>
</tbody>
</table>

Anomalies are gone!

- Minimal Redundancy
- Fast Updates
- Precise Deletion
Functional Dependencies (FD)
What is a Functional Dependency?

Formally:

**Definition** \( A_1, ..., A_m \rightarrow B_1, ..., B_n \) holds in \( R \) if:

\[
\forall t, t' \in R, \\
(t.A_1 = t'.A_1 \land ... \land t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \land ... \land t.B_n = t'.B_n)
\]

<table>
<thead>
<tr>
<th>( R )</th>
<th>( A_1 )</th>
<th>...</th>
<th>( A_m )</th>
<th>( B_1 )</th>
<th>...</th>
<th>( B_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t' )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if \( t, t' \) agree here then \( t, t' \) agree here
What is a Functional Dependency?

Informally:

An FD holds when some attributes imply other attributes
What is a Functional Dependency?

SSN -> Name ?

SSN -> Name, City ?

SSN -> Name, City, PhoneNumber ?
What is a Functional Dependency?

SSN -> Name ?  
Yes

SSN -> Name, City ?

SSN -> Name, City, PhoneNumber ?

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-123-4567</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-890-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>626-246-8024</td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>
What is a Functional Dependency?

SSN -> Name ?
Yes

SSN -> Name, City ?
Yes

SSN -> Name, City, PhoneNumber ?

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-123-4567</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-890-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>626-246-8024</td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>
What is a Functional Dependency?

SSN -> Name ?  Yes

SSN -> Name, City ?  Yes

SSN -> Name, City, PhoneNumber ?  No
What is a Functional Dependency?

SSN -> Name ?
Yes

SSN -> Name, City ?
Yes

SSN -> Name, City, PhoneNumber ?
No

Looks familiar?

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-123-4567</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-890-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>626-246-8024</td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>
What is a Functional Dependency?

SSN -> Name ?
Yes

SSN -> Name, City ?
Yes

SSN -> Name, City, PhoneNumber ?
No

Looks familiar?

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>
Finding FDs

Could be mapped from data… But usually, FDs should be established from prior knowledge about the data.

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-123-4567</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-890-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>626-246-8024</td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>

SSN -> Name

Name -> SSN
Finding FDs

Could be mapped from data… But usually, FDs should be established from prior knowledge about the data.

SSN -> Name ✔
Name -> SSN true for now…
Closure Algorithm

Goal: We want everything that an attribute/set of attributes determine

Observation:

If we have $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$
Closure Algorithm

Goal: We want everything that an attribute/set of attributes determine

Observation:

If we have A -> B and B -> C, then A -> C

So really, A -> B and C

Repeat until X doesn’t change do:
if B₁, ..., Bₙ → C is a FD and B₁, ..., Bₙ are all in X
then add C to X.
Closure Algorithm

Goal: We want everything that an attribute/set of attributes determine

Observation:

If we have A -> B and B -> C, then A -> C

So really, A -> B and C

Formal notation is \( \{A\}^+ = \{A, B, C\} \)
Goal: We want everything that an attribute/set of attributes determine

Observation:

If we have $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$

So really, $A \rightarrow B$ and $C$

Formal notation is $\{A\}^+ = \{A, B, C\}$

Since the closure of $A$ is all attributes, $A$ is a superkey
We call an attribute that determines all other attributes in a schema to be a superkey.

If it is the smallest set of attributes (in terms of cardinality) that does this we call that set a minimal key or just key.
Anomalies

X -> Y in your table schema implies an anomaly UNLESS X is a (super)key

We deal with this by normalizing the schema (i.e. ripping apart tables until these anomalies are gone)
Boyce-Codd Normal Form (BCNF)
What is a “Normal Form”?

Goal of normal forms is to promote consistency, speed, ease of use, etc.

1st Normal Form: Tables are flat

2nd Normal Form: Obsolete

**BCNF: No bad FDs**

3rd Normal Form: See textbook for more details
**Definition.** A relation $R$ is in BCNF if:

Whenever $X \rightarrow B$ is a non-trivial dependency, then $X$ is a superkey.

$\Rightarrow$

**Definition.** A relation $R$ is in BCNF if:

$\forall X$, either $X^+ = X$ or $X^+ = [\text{all attributes}]$
Practical Tips

Normalization is great for promoting consistency about current states.

Fully normalized data can be hindering (think about joins). Denormalizing can bring back redundancy but improve performance in some cases.