**Functional Dependencies** are building blocks that enable the analysis of data redundancies, and the elimination of anomalies caused by them (through the process of normalization).
Example

• Convert to relations:
  - Students(Id, Name)  
  - Advises(StudentId, AdvisorId)  
  - Advisors(Id, Name)  
  - Favorite(StudentId, AdvisorId)

• We perversely decide to convert Students, Advises, and Favorite into one relation.
  – Students(Id, Name, AdvisorId, AdvisorName, FavoriteAdvisorId)
Example of a Bad Relation

Students(Id, Name, AdvisorId, AdvisorName, FavoriteAdvisorId)

• If you know a student's Id, can you determine the values of any other attributes?
  – Name and FavoriteAdvisorId.

  ![Id → Name](Id → Name)
  ![Id → FavoriteAdvisorId](Id → FavoriteAdvisorId)

  ![AdvisorId → AdvisorName](AdvisorId → AdvisorName)

• Can we say $\text{Id} \rightarrow \text{AdvisorId}$?
  – NO! Id is not a key.

• What is the key for the Students?
  – {Id, AdvisorId}

• Why is this relation “bad”?
  – Parts of the key determine other attributes.
Motivation for Functional Dependencies

• Reason about constraints on attributes in relational designs.
• Procedurally determine the keys of a relation.
• Detect when a relation has redundant information.
• Improve database designs systematically using normalization.
Relational Schema Design

Conceptual Model:

Relational Model: plus FD's

Normalization:
Eliminates anomalies
Definition of Functional Dependency

• If \( t \) is a tuple in a relation \( R \) and \( A \) is an attribute of \( R \), then \( t_A \) is the value of attribute \( A \) in tuple \( t \).

• The FD \( \text{AdvisorId} \rightarrow \text{AdvisorName} \) holds in \( R \) if in every instance of \( R \), for every pair of tuples \( t \) and \( u \)
  
  \[
  \text{if } t_{\text{AdvisorId}} = u_{\text{AdvisorId}}, \text{ then } t_{\text{AdvisorName}} = u_{\text{AdvisorName}}
  \]
Definition of Functional Dependency

- $X \rightarrow A$ is an assertion about a relation $R$ that whenever two tuples of $R$ agree on all the attributes of $X$, then they must also agree on the attribute $A$.
  - Say "$X \rightarrow A$ holds in $R$."

- A *functional dependency* (FD) on a relation $R$ is a statement
  - If two tuples in $R$ agree on attributes $A_1, A_2, \ldots, A_n$ then they agree on attribute $B$.
  - Notation: $A_1 A_2 \ldots A_n \rightarrow B$

- FD says that for every pair of tuples $t$ and $u$ in any instance of $R$, if $t_{A_1} = u_{A_1}$ and $t_{A_2} = u_{A_2}$ and \ldots $t_{A_n} = u_{A_n}$, then $t_B = u_B$.

- The set of attributes $A_1, A_2, \ldots A_n$ functionally determine $B$.

- An FD is a constraint on a single relation schema. It must hold on every instance of the relation.

- You cannot deduce an FD from a relation instance.
A functional dependency is a constraint between two sets of attributes in a relation.

An attribute or set of attributes X is said to functionally determine another attribute Y (written $X \rightarrow Y$) if and only if each X value is associated with at most one Y value. Customarily we call X determinant set and Y a dependent set.

So if we are given the value of X we can determine the value of Y.
Examples of FDs

What FDs can we assert for the relation

Courses(Number, DeptName, CourseName, Classroom, Enrollment)

<table>
<thead>
<tr>
<th>Number</th>
<th>DeptName</th>
<th>CourseName</th>
<th>Classroom</th>
<th>Enrollment</th>
</tr>
</thead>
<tbody>
<tr>
<td>4604</td>
<td>CS</td>
<td>Databases</td>
<td>TORG 1020</td>
<td>45</td>
</tr>
<tr>
<td>4604</td>
<td>Dance</td>
<td>Tree Dancing</td>
<td>Drillfield</td>
<td>45</td>
</tr>
<tr>
<td>4604</td>
<td>English</td>
<td>The Basis of Data</td>
<td>Williams 44</td>
<td>45</td>
</tr>
<tr>
<td>2604</td>
<td>CS</td>
<td>Data Structures</td>
<td>MCB 114</td>
<td>100</td>
</tr>
<tr>
<td>2604</td>
<td>Physics</td>
<td>Dark Matter</td>
<td>Williams 44</td>
<td>100</td>
</tr>
</tbody>
</table>

- Number DeptName → CourseName
- Number DeptName → Classroom
- Number DeptName → Enrollment
- Number DeptName → CourseName Classroom Enrollment

- Is Number → Enrollment an FD?
Example

Drinkers(name, addr, beersLiked, manf, favBeer).

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Bud</td>
<td>A.B.</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>WickedAle</td>
<td>Pete’s</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Bud</td>
<td>A.B.</td>
<td>Bud</td>
</tr>
</tbody>
</table>

Reasonable FD’s to assert:

1. name \(\rightarrow\) addr
2. name \(\rightarrow\) favBeer
3. beersLiked \(\rightarrow\) manf
## Example

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Bud</td>
<td>A.B.</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>WickedAle</td>
<td>Pete's</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Bud</td>
<td>A.B.</td>
<td>Bud</td>
</tr>
</tbody>
</table>

- **name** -> **addr**
- **beersLiked** -> **manf**
- **name** -> **favBeer**
FDs With Multiple Attributes

• No need for FDs with > 1 attribute on right.
  – But sometimes convenient to combine FD’s as a shorthand.
  – FDs: name -> addr and name -> favBeer become name -> addr favBeer

• > 1 attribute on left may be essential.
  – Example: bar beer -> price
Use of Functional Dependencies

• We use functional dependencies to:
  – test relations to see if they are legal under a given set of functional dependencies.
    • If a relation $R$ is legal under a set $F$ of functional dependencies, we say that $R$ satisfies $F$.
  – specify constraints on the set of legal relations
    • We say that $F$ holds on $R$ if all legal relations on $R$ satisfy the set of functional dependencies $F$.

• Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
Where do FDs come from?

- “Keyness” of attributes.
- Domain and application constraints.
- Real world constraints, e.g.,
  ProfessorID Time → Classroom
Keys of Relations

- FDs allow us to formally define keys.
- A set of attributes \( \{A_1, A_2, \ldots A_n\} \) is a key for a relation \( R \) if:
  - **Uniqueness** \( \{A_1, A_2, \ldots A_n\} \) functionally determine all the other attributes of \( R \) and
  - **Minimality** no proper subset of \( \{A_1, A_2, \ldots A_n\} \) functionally determines all the other attributes of \( R \).

- A **superkey** is a set of attributes that has the uniqueness property but is not necessarily minimal.

- Note E/R keys have no requirement for minimality, as for relational keys.
Example

Drinkers(name, addr, beersLiked, manf, favBeer).

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
<th>manf</th>
<th>favBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Bud</td>
<td>A.B.</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>WickedAle</td>
<td>Pete’s</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Bud</td>
<td>A.B.</td>
<td>Bud</td>
</tr>
</tbody>
</table>

• \{name, beersLiked\} is a superkey because together these attributes determine all the other attributes.
  – name -> addr favBeer
  – beersLiked -> manf
Example, Cont.

• \{\text{name}, \text{beersLiked}\} is a \textbf{key} because neither \{\text{name}\} nor \{\text{beersLiked}\} is a superkey.
  – name doesn’t \(\rightarrow\) manf;
  – beersLiked doesn’t \(\rightarrow\) addr.

• In this example, there are no other keys, but lots of superkeys.
  – Any superset of \{\text{name}, \text{beersLiked}\}. 
Example of Keys

• What is the key for
  – Courses(Number, DeptName, CourseName, Classroom, Enrollment)?
• The key is \{Number, DeptName\}.
  – These attributes functionally determine every other attribute.
  – No proper subset of \{Number, DeptName\} has this property.

• What is the key for
  – Teach(Number, DepartmentName, ProfessorName, Classroom)?
• The key is \{Number, DepartmentName\}.
  – Why?
Where Do Keys Come From?

- We could simply assert a key $K$. Then the only FD’s are $K \rightarrow A$ for all attributes $A$, and $K$ turns out to be the only key obtainable from the FD’s.

- We could assert FD’s and deduce the keys by systematic exploration.
Keys in the Conversion from E/R to Relational Designs

• If the relation comes from an entity set, the key attributes of the relation are precisely the key attributes of the entity set.
Keys in the Conversion from E/R to Relational Designs

• If the relation comes from a binary relationship $R$ between entity sets $E$ and $F$:

  – $R$ is many-many: key attributes of the relation are the key attributes of $E$ and of $F$.

  – $R$ is many-one from $E$ to $F$: key attributes of the relation are the key attributes of $E$.

  – $R$ is one-one: key attributes of the relation are the key attributes of $E$ or of $F$. 
Keys in the Conversion from E/R to Relational Designs

• If the relationship R is multi-way, we need to reason about the FDs that R satisfies.
  – There is no simple rule.
  – If R has an arrow towards entity set E, at least one key for the relation for R excludes the key for E.
FD’s From “Physics”

• While most FD’s come from E/R keyness and many-one relationships, some are really physical laws.

• Example: “no two courses can meet in the same room at the same time” tells us: hour room -> course.
Example

• Branch

<table>
<thead>
<tr>
<th>branchname</th>
<th>loan</th>
<th>customer</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mall St</td>
<td>17</td>
<td>Jones</td>
<td>1000</td>
</tr>
<tr>
<td>Logan</td>
<td>23</td>
<td>Smith</td>
<td>2000</td>
</tr>
<tr>
<td>Queen</td>
<td>15</td>
<td>Hayes</td>
<td>1500</td>
</tr>
<tr>
<td>Mall St</td>
<td>14</td>
<td>Jackson</td>
<td>1500</td>
</tr>
<tr>
<td>King George</td>
<td>93</td>
<td>Curry</td>
<td>500</td>
</tr>
<tr>
<td>Queen</td>
<td>25</td>
<td>Glenn</td>
<td>2500</td>
</tr>
<tr>
<td>Andrew</td>
<td>10</td>
<td>Brooks</td>
<td>2500</td>
</tr>
<tr>
<td>Logan</td>
<td>30</td>
<td>Johnson</td>
<td>750</td>
</tr>
</tbody>
</table>

• Is Loan → Customer a valid FD?
  • Loan → Customer Amount?
  • Loan → Branchname?
  • Loan → Customer Branchname Amount?
  • Loan Branchname → Amount?
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c2</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c2</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c3</td>
</tr>
</tbody>
</table>

- A → B
- C → B
Rules for Manipulating FDs

- Learn how to reason about FDs.
- Define rules for deriving new FDs from a given set of FDs.
- Next class: use these rules to remove “anomalies” from relational designs.
- **Example:** A relation R with attributes A, B, and C, satisfies the FDs A → B and B → C. What other FDs does it satisfy?
  - A → C
- What is the key for R?
  - A, because A → B and A → C
Equivalence of FDs

• An FD $F$ follows from a set of FDs $T$ if every relation instance that satisfies all the FDs in $T$ also satisfies $F$.

• $A \rightarrow C$ follows from $T = \{A \rightarrow B, B \rightarrow C\}$

• Two sets of FDs $S$ and $T$ are equivalent if each FD in $S$ follows from $T$ and each FD in $T$ follows from $S$.

• $S = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$ and $T = \{A \rightarrow B, B \rightarrow C\}$ are equivalent.

• These notions are useful in deriving new FDs from a given set of FDs.
Inference Rules for FDs

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

Is equivalent to

\[ A_1, A_2, \ldots, A_n \rightarrow B_1 \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_2 \]
\[ \ldots \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_m \]

Splitting rule and Combing rule
Splitting and Combining FDs

- Can we split and combine left hand sides of FDs?

> For the relation Courses is the FD

\[
\text{Number} \leftarrow \text{DeptName} \rightarrow \text{CourseName}
\]

equivalent to the set of FDs

\[
\{\text{Number} \rightarrow \text{CourseName}, \text{DeptName} \rightarrow \text{CourseName}\}
\]?

- No!
Triviality of FDs

An FD $A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m$ is

- **trivial** if the $B$’s are a subset of the $A$’s,
  \[ \{B_1, B_2, \ldots B_n\} \subseteq \{A_1, A_2, \ldots A_n\} \]

- **non-trivial** if at least one $B$ is not among the $A$’s,
  \[ \{B_1, B_2, \ldots B_n\} - \{A_1, A_2, \ldots A_n\} \neq \emptyset \]

- **completely non-trivial** if none of the $B$’s are among the $A$’s, i.e.,
  \[ \{B_1, B_2, \ldots B_n\} \cap \{A_1, A_2, \ldots A_n\} = \emptyset. \]

- **Trivial dependency rule:** The FD $A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m$ is equivalent to the FD $A_1 A_2 \ldots A_n \rightarrow C_1 C_2 \ldots C_k$, where the $C$’s are those $B$’s that are not $A$’s, i.e.,
  \[ \{C_1, C_2, \ldots, C_k\} = \{B_1, B_2, \ldots, B_m\} - \{A_1, A_2, \ldots, A_n\}. \]

- What good are trivial and non-trivial dependencies?
  - Trivial dependencies are always true.
  - They help simplify reasoning about FDs.