The Big Question

- Design
  - Physical (base level tables)
  - Logical (user views)
- How good is our design?
- How do we measure this anyway?

Some Guidelines

- Semantics of the attributes
- Reducing the redundant values in tuples
- Reducing NULL values in tuples
- Disallowing the possibility of generating spurious tuples
Semantics of the Relation Attributes

- Each tuple (record) should represent one entity or relationship instance.
- Attributes of different entities should *not* mix together in the same relation.
- *Only* foreign keys should be used to refer to other entities.
- Entity and relationship attributes should be kept as apart as possible.

Semantics of the Relation Attributes

- Design your database schema such that it can be explained easily relation by relation.
- The semantics of each relation should be easy to interpret. (This is an informal way to measure the “goodness” of your design.)

An Example of Good Design

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>ID</th>
<th>NAME</th>
<th>DEPT</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT</th>
<th>ID</th>
<th>NAME</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>ID</th>
<th>NAME</th>
<th>DEPT</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT</th>
<th>ID</th>
<th>NAME</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Example of Good Design

Guideline 1

- Design relation schema so that it is easy to explain its meaning
- Do not combine attributes from multiple entity types and relationship types into a single relation
- Example of violating Guideline 1: Figure 15.3

Examples of Not-So-Good Design
Redundant Information in Tuples and Update Anomalies

- Mixing attributes of multiple entities may cause problems.
- Information stored redundantly may waste space.
- Problems with update anomalies
  - Insertion
  - Modification
  - Deletion

Guideline 2

- Design base relation schemas so that no update anomalies are present in the relations
- If any anomalies are present:
  - Note them clearly
  - Make sure that the programs that update the database will operate correctly

Examples of Not-So-Good Design

- What about new dept. w/o any employees?
Examples of Not-So-Good Design

NULL Values in Tuples
- May group many attributes together into a “fat” relation
  - Can end up with many NULLs
- Problems with NULLs
  - Wasted storage space
  - Problems understanding meaning

Guideline 3
- Avoid placing attributes in a base relation whose values may frequently be NULL
- If NULLs are unavoidable:
  - Make sure that they apply in exceptional cases only, not to a majority of tuples
Spurious Tuples

- Poor design can lead to spurious tuples when performing certain join operations.
- The "lossless join" property is used to guarantee meaningful results for join operations.

Guideline 4

- Design relation schemas to be joined with equality conditions on attributes that are appropriately related
  - Guarantees that no spurious tuples are generated
- Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations
Spurious Tuples

<table>
<thead>
<tr>
<th>ENH</th>
<th>PRJNUM</th>
<th>HOURS</th>
<th>ENAME</th>
<th>PLOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>1</td>
<td>20.5</td>
<td>Product X</td>
<td>Seattle</td>
</tr>
<tr>
<td>12345678</td>
<td>2</td>
<td>7.5</td>
<td>Product Y</td>
<td>Seattle</td>
</tr>
<tr>
<td>12345678</td>
<td>3</td>
<td>20.0</td>
<td>Product Z</td>
<td>Houston</td>
</tr>
<tr>
<td>12345678</td>
<td>4</td>
<td>10.0</td>
<td>Product A</td>
<td>Seattle</td>
</tr>
<tr>
<td>12345678</td>
<td>5</td>
<td>20.0</td>
<td>Product X</td>
<td>Seattle</td>
</tr>
<tr>
<td>12345678</td>
<td>6</td>
<td>15.0</td>
<td>Product Z</td>
<td>Houston</td>
</tr>
<tr>
<td>12345678</td>
<td>7</td>
<td>20.0</td>
<td>Product X</td>
<td>Seattle</td>
</tr>
<tr>
<td>12345678</td>
<td>8</td>
<td>15.0</td>
<td>Product Z</td>
<td>Houston</td>
</tr>
<tr>
<td>12345678</td>
<td>9</td>
<td>20.0</td>
<td>Communication</td>
<td>Seattle</td>
</tr>
<tr>
<td>12345678</td>
<td>10</td>
<td>15.0</td>
<td>Communication</td>
<td>Houston</td>
</tr>
</tbody>
</table>

Summary and Discussion of Design Guidelines

- Anomalies cause redundant work to be done
- Waste of storage space due to NULLs
- Difficulty of performing operations and joins due to NULL values
- Generation of invalid and spurious data during joins
Functional Dependencies

- Formal tool for analysis of relational schemas
- Enables us to detect and describe some of the above-mentioned problems in precise terms
- Theory of functional dependency

Definition of Functional Dependency

- Constraint between two sets of attributes from the database
  Definition. A functional dependency, denoted by \( X \rightarrow Y \), between two sets of attributes \( X \) and \( Y \) that are subsets of \( R \) specifies a constraint on the possible tuples that can form a relation state \( r \) of \( R \). The constraint is that, for any two tuples \( t_i \) and \( t_j \) in \( r \) that have \( t_i[X] = t_j[X] \), they must also have \( t_i[Y] = t_j[Y] \).
- Property of semantics or meaning of the attributes
- Legal relation states
  - Satisfy the functional dependency constraints

Definition of Functional Dependency (cont.)

- A set of attributes \( X \) functionally determines a set of attributes \( Y \) if the value of \( X \) uniquely determines the values of \( Y \)
- If two tuples have the same \( X \) values, then they must have the same \( Y \) values
- A set of attributes \( Y \) is functionally dependent on the set of attributes \( X \).
Functional Dependencies

- For instance:
  - SSN->ENAME
  - PNUMBER->\{PNAME, PLOCATION\}
  - \{SSN, PNUMBER\}->HOURS

Functional Dependencies

- Given a populated relation
  - Cannot determine which FDs hold and which do not
  - Can state that FD does not hold if there are tuples that show violation of such an FD

Normalization of Relations

- Takes a relation schema through a series of tests
  - Certify whether it satisfies a certain normal form
  - Proceeds in a top-down fashion

- Normal form tests

Definitions. The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.
Normalization of Relations (cont’d.)

Properties that the relational schemas should have:
- Nonadditive join property
  - Extremely critical
- Dependency preservation property
  - Desirable but sometimes sacrificed for other factors

Normalization of Relations (cont’d.)

Normalization: the process of decomposing unsatisfactory relations by breaking their attributes into smaller ones
Normal form: condition using keys and FDs of a relation to certify whether a relation scheme is in a particular form

First Normal Form

Part of the formal definition of a relation in the basic (flat) relational model
Only attribute values permitted are single atomic (or indivisible) values
Techniques to achieve first normal form
- Remove attribute and place in separate relation
- Expand the key
- Use several atomic attributes
Definitions of Keys and Attributes

Participating in Keys

- Definition of superkey and key
- Candidate key
  - If more than one key in a relation schema
    - One is primary key
    - Others are secondary keys

Second Normal Form

- Based on concept of full functional dependency
  - Versus partial dependency

Definitions: An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R. An attribute is called nonprime if it is not a prime attribute—that is, if it is not a member of any candidate key.

Second normalize into a number of 2NF relations
- Nonprime attributes are associated only with part of primary key on which they are fully functionally dependent
2NF

- Examples:
  - (SSN, Pnumber)->Hours is full.
  - (SSN, Pnumber)->EName is "not" full.