Building Database Applications: Steps

1. Start with a conceptual model
   - "On paper" using certain techniques
     - E-R Model
   - ignore low-level details – focus on logical representation
   - "step-wise refinement" of design with client input
2. Design & implement schema
   - Design and codify (in SQL) the relations/tables
   - Refine the schema – normalization
   - Do physical layout – indexes, etc.
3. Import the data
4. Write applications using DBMS and other tools
   - Many of the hard problems are taken care of by other people (DBMS, API writers, library authors, web server, etc.)
   - DBMS takes care of Query Optimization, Efficiency, etc.

Next:

Database Design Process- How to design a good schema?
Relational Model: Definitions Review

- Relations/tables, Attributes/Columns, Tuples/rows
  - Attribute domains
- Superkey
- Key
  - No two tuples can have the same value in the key attribute
  - Primary key, candidate keys
  - No primary key value can be null
- Referential integrity constraints
  - Foreign key

Next… how to design the schema?

Relational Schema Design

- Logical Level
  - Whether schema has intuitive appeal for users
- Manipulation level
  - Whether it makes sense from an efficiency or correctness point of view

Functional Dependencies and Normal Forms

- Guidelines for database schema design: how to design a “good” schema?
- Example of a COMPANY database: two possible designs to represent Employees and Department information

S1: EMPLOYEE(LNAME,FNAME,SSN,DNO)
    DEPT(DNUM, DNAME, MGRSSN)

S2: EMPDEPT(LNAME,FNAME,SSN,DNUM,DNAME,MGRSSN)

Which one is better?? S1 or S2?

Functional Dependencies and Normal Forms

- Guidelines for database schema design: how to design a “good” schema?
- Example of a COMPANY database: two possible designs to represent Employees and Department information

S1: EMPLOYEE(LNAME,FNAME,SSN,DNO)
    DEPT(DNUM, DNAME, MGRSSN)

S2: EMPDEPT(LNAME,FNAME,SSN,DNUM,DNAME,MGRSSN)

Which one is better?? S1 or S2?
Functional Dependencies and Normal Forms

- Informal methods
  - Rules of thumb, intuitive reasoning, experience
- Formal methods
  - Provable properties
  - Involve concept of Functional Dependencies
  - Develop theoretical model to define what we mean by “good schema”
  - Normal forms are defined in terms of properties of the functional dependencies and link to ‘good’ or ‘bad’ schema design
    - If a relation is in Third Normal Form it is a “good” design etc.

Informal Guidelines: 1

- 1: Try to make user interpretation easy
  - S1: EMP(FNAME, LNAME, SSN)
    WORKS_ON (SSN, PNO)
    PROJECT_LOC(PNO, PLOC)
  - S2: EMP(FNAME, LNAME, SSN, PNO, PLOC)
    - Perhaps S2 has too much information to absorb per tuple?

Informal Guidelines: 2

- Try to reduce redundancy
  - In S2 in previous example, suppose only few projects PLOC is unnecessarily repeated too often
  - On the other hand, S1 repeats SSN in WORKS_ON
    But SSN is a smaller attribute than PLOC (which may be a large string)

Informal Guidelines: 3

- Try to avoid update anomalies
  - Avoid having to search through entire table during update operation
    Insert, delete, update/modify
    Searching using a non-key attribute may require searching the entire table
  - Avoid losing information
    - This is an important criteria
      - efficiency
Informal Guidelines: 4
- Avoid too many NULL values
  - Space is wasted...why is this a problem?
  - Problems occur when using aggregate functions like count or sum
  - NULLs can have different intentions
    - Attribute does not apply
    - Value unknown and will remain unknown
    - Value unknown at present

Informal Guidelines: 5: Lossless Joins & Spurious Tuples
- Split a table into smaller tables (with fewer columns in each)
  - sometimes a better design in our examples
  - How to split?
- Must be able to reconstruct the ‘original’ table
- When reconstructing the “original” data, should not introduce spurious tuples
  - Also called non-additive joins

Informal Guidelines: 6
- Do not lose the dependencies
  - Will define this after we define functional dependencies....
- Informally:
  - Do not lose constraints and business rules when we decompose tables into smaller tables

Example
S1: EMPLOYEE (ENAME, SSN, BDATE, ADDR, DNO)
  DEPT (DNUM, MGRSSN, DNAME)
  WORKS_ON (SSN, PNO, HOURS)
  PROJECT (PNUM, PLOC)

S2: EMP_DEPT(ENAME,SSN,BDATE,ADDR,DNO,DNAME,MGRSSN)
  EMP_PROJ (SSN_PNUM, HOURS, ENAME, PNAME, PLOC)

Both schemas have same attributes....
Problems with S2 ??
Question: Insertion queries in S2
- Consider inserting information “John Smith works in Department 5”
  - i.e., tuple should be inserted into EMP_DEPT
    - < John Smith, 23456789,…,5,Research,111223333>
- Are there any problems you see?

Question
- Create new department in the company with DNO=9 and DNAME= ‘Sales’
- Any problems?

Answer 1: Insertion Anomalies in S2
- What do we need to check in the table to maintain correctness?
- Check data is correct – i.e.,
  - Check that all tuples with DNO=5 have DNAME=Research, MGRSSN=111223333
  - Scan the whole relation since DNO is not a key!

Answer 2: Insertion problems…contd.
- Consider creating a new department in the Company: DNO=9, DNAME = ‘Sales’
- Only one way to do this: create NULL values for employee info
  - But that means NULL value in primary key (SSN)!!
**Deletion**

- What if we delete the last employee in ‘Research’ department
  - Eg. (John Smith, 123456789,….5,’Research’,….)

**Deletion and Modification Anomalies**

- What if we delete the last employee in ‘Research’ department
  - Eg. (John Smith, 123456789,….5,’Research’,….)
  - We lose the information that Dept 5 is Research
- Similar cases for Modification
  - Change Manager SSN of department 5 = change for all Department 5 employees …scan of entire table

**Question**

- Change/update manager SSN of department 5
- Any problems?

**Example: Decomposing a table into smaller tables**

<table>
<thead>
<tr>
<th>ID</th>
<th>Make</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Toyota</td>
<td>Blue</td>
</tr>
<tr>
<td>456</td>
<td>Audi</td>
<td>Blue</td>
</tr>
<tr>
<td>789</td>
<td>Toyota</td>
<td>Red</td>
</tr>
</tbody>
</table>

Original Table S has 3 columns

Decompose into:
- S1: CAR (ID, Make, Color)
- S2: CAR1 (ID, Color)
  - CAR2 (Color, Make)
Question: Is there a problem with this decomposition?

<table>
<thead>
<tr>
<th>S2-Car 1</th>
<th>S2-Car 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>123 Blue</td>
<td>Blue Toyota</td>
</tr>
<tr>
<td>456 Blue</td>
<td>Blue Audi</td>
</tr>
<tr>
<td>789 Red</td>
<td>Red Toyota</td>
</tr>
</tbody>
</table>

Spurious tuples...We do not get back the original table

| 123 Blue Toyota |
| 123 Blue Audi   |
| 456 Blue Toyota |
| 456 Blue Audi   |
| 789 Red Toyota  |

Summary of Problems

- Insertion, Deletion, modification anomalies
- Too many NULLs
- Spurious tuples – called non-additive join
- We need a theory of schema design
  - Functional dependencies and normalization
  - Using functional dependencies define “normal forms” of schema
    - A schema in a “Third Normal Form” will avoid certain anomalies

Normalization

- **Normalization** is a technique for producing relations with desirable properties.

- Normalization decomposes relations into smaller relations that contain less redundancy. This decomposition requires that no information is lost and reconstruction of the original relations from the smaller relations must be possible.

- Normalization is a bottom-up design technique for producing relations. It pre-dates ER modeling and was developed by Codd in 1972 and extended by others over the years.
  - Normalization can be used after ER modeling or independently.
  - Normalization may be especially useful for databases that have already been designed without using formal techniques.
Normalization Motivation

- The goal of normalization is to produce a set of relational schemas $R_1, R_2, \ldots, R_m$ from a set of attributes $A_1, A_2, \ldots, A_n$.
  - Imagine that the attributes are originally all in one big relation $R = \{A_1, A_2, \ldots, A_n\}$ which we will call the **Universal Relation**.
  - Normalization divides this relation into $R_1, R_2, \ldots, R_m$.

Desirable Relational Schema Properties

- Relational schemas that are well-designed have several important properties:
  1) The most basic property is that relations consist of attributes that are logically related. The attributes in a relation should belong to only one entity or relationship.
  2) **Lossless-join property** ensures that the information decomposed across many relations can be reconstructed using natural joins.
  3) **Dependency preservation property** ensures that constraints on the original relation can be maintained by enforcing constraints on the normalized relations.
  4) Avoid update anomalies

Functional Dependencies

- Functional dependencies represent constraints on the values of attributes in a relation and are used in normalization.

- A **functional dependency** (abbreviated FD) is a statement about the relationship between attributes in a relation. We say a set of attributes $X$ functionally determines an attribute $Y$ if given the values of $X$ we always know the only possible value of $Y$.
  - Notation: $X \rightarrow Y$
  - $X$ functionally determines $Y$
  - $Y$ is functionally dependent on $X$

  Example:
  - $\text{eno} \rightarrow \text{ename}$
  - $\text{eno, pno} \rightarrow \text{hours}$

Notation for Functional Dependencies

- A functional dependency has a left-side called the **determinant** which is a set of attributes, and one attribute on the right-side.

$$ \text{eno, pno} \rightarrow \text{hours} $$

  - Strictly speaking, there is always only one attribute on the RHS, but we can combine several functional dependencies into one:
  - $\text{eno, pno} \rightarrow \text{hours}$
  - $\text{eno, pno} \rightarrow \text{resp}$

  $$ \text{eno, pno} \rightarrow \text{hours, resp} $$

  - Remember that this is really short-hand for two functional dependencies.
**Why the Name "Functional" Dependencies?**

- Functional dependencies get their name because you could imagine the existence of some function that takes in the parameters of the left-hand side and computes the value on the right-hand side of the dependency.

- Example: $\text{eno, pno} \rightarrow \text{hours}$
  
  ```java
  int f(String eno, String pno)
  {
    // Do some lookup...
    return hours;
  }
  ```

- Remember that no such function exists, but it may be useful to think of FDs this way.

**Question**

- We have attributes (A1, A2, A3, A4) in a table
- We have functional dependencies
  
  $A1 \rightarrow A2$
  
  $A1 \rightarrow A3$
  
  $A1 \rightarrow A4$

**Next: Functional Dependencies & Normal Forms**

- Normalization requires decomposing a relation into smaller tables
- Normal forms are properties of relations
- We say a relation is in xNF if its attributes satisfy certain properties
  
  - Properties formally defined using functional dependencies
  - For example, test the relation to see if it is in 3NF
  - If not in 3NF, then change design...how?

**How to go about designing a good schema?**

- How to create a 3NF database schema? (i.e., a good design)?
- Ad-hoc approach
  
  - Create relations intuitively and hope for the best!
- Formal method – procedure
  
  - Start with single relation with all attributes
  - Systematically decompose relations that are not in the desired normal form
  - Repeat until all tables are in desired normal form
  - Can decomposition create problems if we are not careful?
    
    - Yes: (i) Spurious tuples and (ii) lost dependencies

- Can we automate the decomposition process...

  **Input:** Set of attributes and their functional dependencies
  
  **Output:** A 'good' schema design