Today’s Agenda

- Quiz (finally!)
- More on Database Design
  - Ternary relationships
  - Normalization
Ternary versus Binary Relationships

- These two schemas are not equivalent!
- When would we use a ternary relationship set?
- When would we use three binary relationship sets?

This ternary relationship set means that a supplier must be authorized to supply a particular part to a particular project.

For example:
- "office depot" can supply "pencils" to "project 112"
- "office max" can supply "paper" to "project 115"
- but this does not imply "office max" can supply "paper" to "112"
Ternary versus Binary Relationships

This *ternary* relationship set means that a supplier must be authorized to supply a particular part to a particular project. For example:
- "office depot" can supply "pencils" to "project 112"
- "office max" can supply "paper" to "project 115"
- but this does not imply "office max" can supply "paper" to "112"

Ternary versus Binary Relationships

The three binary relationship sets each represent something *distinct*
- a supplier can be provide certain products ("office max can provide pencils")
- a project can require certain products ("112 requires pencils")
- a supplier can be authorized to use a certain supplier ("112 is authorized to use office max")
- therefore, we might assume that office-max supplies pencils to 112
Ternary versus Binary Relationships

The three binary relationship sets each represent something distinct:

- A supplier can be provide certain products (office max can provide pencils)
- A project can require certain products (112 requires pencils)
- A supplier can be authorized to use a certain supplier (112 is authorized to use office max)
- Therefore, we might assume that office-max supplies pencils to 112

Representing Ternary as Binary Relationships

We can use binary relationships for ternary ones if we:

... use a new entity to represent the ternary relationship
Normalization

“Normalization” is the process of replacing a table with two or more tables

• For example, consider this schema:

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Dept</th>
<th>DeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Ali</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A12</td>
<td>Eric</td>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

• Versus these:

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>DeptID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Ali</td>
<td>12</td>
</tr>
<tr>
<td>A12</td>
<td>Eric</td>
<td>10</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DeptID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

• Which schema do you think is better? Why?

Normalization Issues

• The EmpDept schema combines two different concepts
  – Employee information, together with
  – Department information

• What is wrong with this?
  – If we separate the two concepts we could save space but some queries would run slower (Joins)
  – If we combine the two ideas we have redundancy but some queries would run faster (no Joins)

So we have a tradeoff …

Redundancy has a side effect: “anomalies”
Types of Anomalies

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Dept</th>
<th>DeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Ali</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A12</td>
<td>Eric</td>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

“Update Anomaly”: If the CS department changes its name, we must change multiple rows in EmpDept.

“Insertion Anomaly”: If a department has no employees, where do we store its ID and name?

“Deletion Anomaly”: If A12 quits, the information about the HR department will be lost.

- These are in addition to redundancy in general
  - For example, the department name is stored multiple times

Using NULL values can help insertion and deletion anomalies

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Dept</th>
<th>DeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Ali</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

Using NULL values
- But NULL values have their own issues
  - They make aggregate operators harder to use
  - Not always clear what NULL means (does not apply, unknown, know but missing, see Melton podcast!)
  - May need outer joins instead of ordinary joins
  - In this case, EID is a primary care, and so it cannot contain a NULL value!
- They don’t address update anomalies or redundancy issues
Decomposition

Normalization involves decomposing (partitioning) the table into separate tables:

- Check to see if redundancy still exists (… repeat)

The key to understanding when and how to decompose schemas is through \(\text{“functional dependencies”}\)

- which generalizes the notion of keys

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>DeptID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Ali</td>
<td>12</td>
</tr>
<tr>
<td>A12</td>
<td>Eric</td>
<td>10</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
</tr>
</tbody>
</table>

The “normalized” version of EmpDept

<table>
<thead>
<tr>
<th>DeptID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

Keys

Because EID is a key:

- If two rows have the same EID value, then they have the same value for every other attribute
- Thus given an EID value, the other values are “determined”

A Key is like a “function”:

- \( f : \text{EID} \rightarrow \text{Name} \times \text{Dept} \times \text{DeptName} \)
- E.g., \( f(A01) = \langle \text{Ali}, 12, \text{CS} \rangle \)

- Recall functions always return the same value for a given value
Functional Dependencies

- We say that EID “functionally determines” all other attributes.
- This relationship among attributes is called a “Functional Dependency” (FD).
- We write FDs as:
  \[ EID \rightarrow Name, Dept, DeptName \]
  or
  \[ EID \rightarrow Name, EID \rightarrow Dept, EID \rightarrow DeptName \]

FDs that are not implied by keys

- Is Name → Dept a functional dependency?
  - No, e.g., <Eric, 10> and <Eric, 12>
- Is Dept → DeptName a functional dependency?
  - Yes in this table it is.
  - In general, it would be expected that departments only have one name.
Functional Dependencies

• For sets A and B of attributes in a relation, we say that A (functionally) determines B … or A → B is a Functional Dependency (FD)
  – if whenever two rows agree on A they also agree on B

• An FD defines a function in the “mathematical sense”

• There are two special kinds of FDs:
  “Key FDs” of the form X → A where X contains a key (X is called a superkey)
  “Trivial FDs” of the form A → B such that A ⊆ B
  – … e.g.,  (Name, Dept) → Dept
  – these are boring but become important later

Likely functional dependencies:
  ssn → name
  account → balance

Unlikely functional dependencies:
  – date → trasactionid
  – checkamt -> checknumber
Enforcing Functional Dependencies

- For the table
  \[ \text{Emp}(eid, \text{name, dept, deptname}) \]

  There is an FD from from \( \text{dept} \rightarrow \text{deptname} \)

  Although \( eid \) is the key for this table …

  … is it still possible for there to be two names for the same department?

  YES!

Every Key Implies a Set of FDs

- For the table
  \[ \text{Emp}(eid, \text{name, dept, deptname}) \]

  We have the following FDs based on ssn being a key:

  \[
  \begin{align*}
  \text{eid} & \rightarrow \text{name} \\
  \text{eid} & \rightarrow \text{dept} \\
  \text{eid} & \rightarrow \text{deptname}
  \end{align*}
  \]

  - Each key implies a set of functional dependencies from the key to the non-key attributes
Functional Dependencies and Keys

- Given a table $R$ with attributes $a$ and $b$ together forming a key, the following FDs are implied

  - Given $R(a, b, c, d, e)$
    
    $ab \rightarrow c$
    
    $ab \rightarrow d$
    
    $ab \rightarrow e$

  Which we can also write as $ab \rightarrow cde$

Functional Dependencies May Suggest Keys

- If we know these FDs:

  $ssn \rightarrow name$

  $ssn \rightarrow hiredate$

  $ssn \rightarrow phone$

- then $ssn$ is a key for a table with these attributes:

  Employee($ssn$, name, hiredate, phone)
What are the key and non-trivial FDs?

Customer(CustID, Address, City, Zip, State)

Enrollment(StdntID, ClassID, Grade, InstrID, StdntName, InstrName)

The Non-Trivial FDs

Customer(CustID, Address, City, Zip, State)

Enrollment(StdntID, ClassID, Grade, InstrID, StdntName, InstrName)

Which of these will be enforced?
Non-Trivial Functional Dependencies

• The FDs that are not enforced by the DBMS lead to both redundancy and anomalies (only keys are enforced)

• Not all redundancy is covered by FDs
  - Emp(ssn, name, salary, birthdate)
  - Emp(ssn, name, address)
  - name stored redundantly, and same employee can have more than one name

• Cannot be determined from the instance (instead, based on application semantics)
  - We can determine what is not an FD
  - DB data mining approaches infer “FDs” (i.e., association rules)

Example Decomposition based on FDs

• For this table
  - Emp(ssn, name, birthday, address, dnum, dname, dmgr)

• We can move the non-trivial FDs into their own table with dnum as the key:
  - Dept(dnum, dname, dmgr)

• The Emp table becomes:
  - Emp(ssn, name, birthday, address, dept)

• … and Emp.dept is now a foreign key to Dept.dnum
Normalization Based on FDs

- Identify all all the FDs
  - FDs implied by the keys
  - FDs not implied by the keys (the “troublesome” ones)
- Generate one or more new tables from the FDs not implied by the keys
  - Each new tables should only have FDs implied by the key
- Remove the attributes from original table that are functionally dependent on “troublesome” FDs
- Specify appropriate foreign keys to these new tables

Reasoning about Functional Dependencies

EmpDept(EID, Name, DeptID, DeptName)

- Two natural FDs are
  - EID → DeptID and DeptID → DeptName
- These two FDs imply EID → DeptName
  - If two tuples agree on EID, then by EID → DeptID they agree on DeptID …
  - … and if they agree on DeptID, then by DeptID → DeptName they agree on DeptName

- The set of FDs implied by a given set F of FDs is called the closure of F … which is denoted F+
### Armstrong’s Axioms

- The closure $F^+$ of $F$ can be computed using these axioms
  - **Reflexivity**: If $X \supseteq Y$, then $X \rightarrow Y$
  - **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any $Z$
  - **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

- Repeatedly applying these rules to $F$ until we no longer produce any new FDs results in a *sound and complete* inference procedure ...

- **Soundness**
  - Only FDs in $F^+$ are generated when applied to FDs in $F$

- **Completeness**
  - Repeated application of these rules will generate all FDs in $F^+$

---

### Finding Keys

We can determine if a set of attributes $X$ is a key for relation $R$ by computing $X^+$ as follows

*Compute $X^+$ from $X$*

1. let $X^+ = \{X\}$
2. repeat until there is no change in $X^+$
3. {
4.   if $Y \rightarrow Z$ is an FD and $Y \subseteq X^+$ then
5.     $X^+ = X^+ \cup Z$
6.   }
7. return $X^+$

- Let the set of attributes of $R$ be $A$
- $X$ is a key for $R$ if and only if $X^+ = A$
Example

- Given the schema \( R(A, B, C, D, E) \) such that
  - \( BC \rightarrow A \)
  - \( DE \rightarrow C \)
- Find the keys of this schema, besides \( A \) …

- Start with \( BC \rightarrow A \) as one example
  - \( BC \) determines \( A \) is given
  - \( A \rightarrow ABCDE \) because \( A \) is a key
  - \( BC \rightarrow ABCDE \) by transitivity
  - Thus, \( BC \) is a key!

- You should understand the axioms and the algorithm … they will come in handy when normalizing

Redundancy and Functional Dependencies

- Example schema

  EmpDept(\( EID, Name, Dept, DeptName \))
  Assigned(\( EmpID, JobID, EmpName, Percent \))
  Enrollment(\( StdntID, ClassID, Grade, InstrID, StdntName, InstrName \))

- Note that every non-key FD is associated with some redundancy
- Our game plan is to use non-key and non-trivial FDs to decompose any relation into a form that has no redundancy …
- … resulting in a so-called “Normal Form”
Boyce-Codd Normal Form (BCNF)

- A relation is in “Boyce-Codd Normal Form” if all of its FDs are either
  - Trivial FDs (e.g., \(AB \rightarrow A\))
  or
  - Key FDs

- Which (if any) of these relations is in BCNF?

  EmpDept (EID, Name, Dept, DeptName)
  Assigned (EmptID, JobID, EmpName, Percent)
  Enrollment (StdntID, ClassID, Grade, InstrID, StdntName, InstrName)

BCNF and Redundancy

BCNF relations have no redundancy caused by FDs

- A relation has redundancy if there is an FD between attributes
- … and there can be repeated entries of data for those attributes

- For example, consider

<table>
<thead>
<tr>
<th>DeptID</th>
<th>DeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

- if the relation is in BCNF, then the FD must be a key FD, and so DeptID must be a key
- implying that any pair such as \(<12, CS>\) can appear only once!
Decomposition into BCNF

- An algorithm for decomposing a relation R with attributes A into a collection of BCNF relations

1. if R is not in BCNF and X → Y is a non-key FD then
2. decompose R into A – Y and XY
3. if A – Y and/or XY is not in BCNF then
4. recursively apply step 1 (to A – Y and/or XY)

Example

Enrollment(StdntID, ClassID, Grade, InstrID, StdntName)

- First use the non-key FD StdntID → StdntName
- … which gives the decomposition
  Enrollment(StdntID, ClassID, Grade, InstrID)
  Student(StdntID, StdntName)

- Now use the non-key FD ClassID → InstrID
- … which gives the decomposition
  Enrollment(StdntID, ClassID, Grade)
  ClassInstructor(ClassID, InstrID)
  Student(StdntID, StdntName)

- All relations are now in BCNF!
Where are we?

- We have accomplished a lot!
  - we began with a relational schema
  - we identified problems with redundancy
  - we used FDs to eliminate those problems with decompositions into BCNF
  - along the way we learned how to identify keys using FDs

- There are two steps left …
  - ensure that the BCNF decompositions do not lose information (are not "lossy")
  - it turns out that in some cases we may lose FDs, but there is a way to deal with this

For Thursday

- Reading
  - Ch 19: Intro, 19.1-19.4

- Be sure to understand:
  - Binary versus ternary relationships
  - Keys, Functional Dependencies, and Boyce-Codd Normal Form (FD)

- Homework
  - Homework 3 due next Thursday
  - Homework 4 and Part 2 of the project coming soon!