Today …

- Normalization

Homework

- HW 2 due
- HW 3 out soon
**Keys revisited**

EmpDept

<table>
<thead>
<tr>
<th>eid</th>
<th>name</th>
<th>dept</th>
<th>dept_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Alice</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>

Because *eid* is a key (... a different take on a key constraint)

- If we know the *eid* value, all other values are known
- If 2 rows had same *eid* value, they have same values for every other attribute
- Thus, given an *eid* value, all other values are “determined”

A key is like a (mathematical) “function”

- a function always returns the same value for a given input
- \( f: \text{eid} \rightarrow \text{name} \times \text{dept} \times \text{dept\_name} \) ... cartesian product of domains
- e.g.: \( f(A01) = \langle \text{Alice}, 12, \text{CS} \rangle \)

We say that *eid* “functionally determines” all other attribute values

- This relationship is called a “functional dependency” (FD)
- And write FDs as:
  - *eid* \( \rightarrow \) name, dept, dept\_name
  - which implies: *eid* \( \rightarrow \) name, *eid* \( \rightarrow \) dept, and *eid* \( \rightarrow \) dept\_name
**Functional Dependencies**

Not all FDs have to be on (implied by) keys

Q: Which of these could be functional dependencies?

- name → dept
- name → dept_name
- dept → dept_name
- dept_name → dept ... YES!
- dept_name → dept ... Maybe (if dept. names are unique)

For attribute sets \( X \) and \( Y \), \( X \rightarrow Y \) is a functional dependency ...

- if whenever two rows agree on \( X \) they also agree on \( Y \)
- if so, we say \( X \) functionally determines \( Y \)

There are three special kinds of FDs ... \( X, Y \) are sets of attributes

- **Key FDs** of the form \( X \rightarrow Y \) where \( X \) contains a key
  - i.e., \( X \) is a superkey
  - the database can enforce these for us

- **Trivial FDs** of the form \( X \rightarrow Y \) such that \( Y \subseteq X \)
  - e.g: name, dept → dept
  - these are “boring”

- **Non-Key, Non-Trivial** FDs
  - The rest: the non-key FDs that aren’t trivial
  - These are the “bad” ones

Like keys, FDs are based on the application semantics
Enforcing functional dependencies

For our table

\[ \text{EmpDept}(\text{eid}, \text{name}, \text{dept}, \text{deptname}) \]

- with key \text{eid}
- and FD \text{dept} \rightarrow \text{deptname}

Q: Although \text{eid} is the key for this table ... is it still possible for there to be 2 names for the same department?

- YES! ... because of the FD from \text{dept} \rightarrow \text{deptname}
- The DBMS can enforce candidate keys, but not non-key, non-trivial FDs

What are possible non-key, non-trivial FDs in this example?

\[ \text{Enrollment}(\text{student_id}, \text{class_id}, \text{instructor_id}, \text{student_name}, \text{instructor_name}) \]

- instructor_id \rightarrow instructor_name
- student_id \rightarrow student_name
Second Normal Form (2NF)

A relation is in 2NF if:

- every non-key attribute is fully dependent on each candidate key
- note that a key (i.e., "prime") attribute is in at least one candidate key

A relation is in 2NF if for every non-trivial FD $X \rightarrow Y$, either:

- $X$ is not a proper subset of a candidate key; or else
- $Y$ contains attributes only from candidate keys (i.e., only prime attributes)

Q: Is this relation in 2NF?

Enrollment(student_id, class_id, instructor_id, student_name, instructor_name)

- No, because of the FD: student_id $\rightarrow$ student_name
- And this FD can’t be enforced by the DBMS (via keys)

Q: Is this decomposition in 2NF?

Enrollment(student_id, class_id, instructor_id, instructor_name)

Student(student_id, student_name)

- Yes, even with the FD: instructor_id $\rightarrow$ instructor_name
- This FD also can’t be enforced by the DBMS (via keys)
Third Normal Form (3NF)

A relation is in 3NF if for every non-trivial FD $X \rightarrow Y$, either:

- $X \rightarrow Y$ is a key FD ($X$ is a superkey); or
- $Y$ is a part of some candidate key for $R$

3NF sometimes defined as 2NF without “transitive dependencies”

- i.e., without FDs of the form $X \rightarrow Y$ where $X$ and $Y$ are non-prime
- similarly, for every non-trivial FD $X \rightarrow Y$ with non-prime $Y$, $X$ is a superkey

Sometimes 3NF is as far as we can go ...

- a notion of “circling” back to (a part of) the key
- and can’t break up the key (without losing information)

Example of a relation in 3NF:

- Location(address, city, state, zip)
- where:
  - address, city, state $\rightarrow$ zip
  - zip $\rightarrow$ state
- zip is a non-prime attribute, but state is a prime attribute
Boyce-Codd Normal Form (BCNF)

A relation is in BCNF if all of its non-trivial FDs are
- **Key FDs** (of the form $X \rightarrow Y$ for superkey $X$)

Are either of these relations in BCNF? (Why or why not...)

- **EmpDept**($eid$, name, dept, dept_name)
- **Assigned**($eid$, pid, emp_name, percent)

BCNF relations have **no redundancy** caused by FDs
- redundancy if there is an FD between attributes
- and there can be repeated entries of data for those attributes

Example BCNF decomposition (based on FDs) for EmpDept:

<table>
<thead>
<tr>
<th>dept</th>
<th>dept_name</th>
<th>eid</th>
<th>name</th>
<th>dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>CS</td>
<td>A01</td>
<td>Alice</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>HR</td>
<td>A12</td>
<td>Eric</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A13</td>
<td>Eric</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A03</td>
<td>Anne</td>
<td>12</td>
</tr>
</tbody>
</table>

- with FDs:
  - dept $\rightarrow$ dept_name
  - eid $\rightarrow$ name, dept
Properties of Decompositions

Basic idea of normalization

Decompose a table using the “bad” FDs $X \rightarrow Y$ by ...

- removing $Y$ from the original table
- creating a table out of $XY$
- making $X$ the new primary key for the $XY$ table

A “good” decomposition is considered:

- **Lossless** ... we can get the original table back
- **Dependency Preserving** ... can still enforce all of the FDs
- in BCNF (if possible) or else 3NF

Checking for Lossless Decompositions ...

- if relation $R(A)$ is decomposed into $R_1(A_1)$ and $R_2(A_2)$
- its lossless iff $A_1 \cap A_2$ contains a key in either $R_1$ or $R_2$

Lossless means we can get back the original relation before decomposition

- if two relations don’t share a key ...
- we could get back more rows than the original