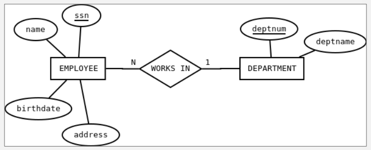
Relational Design Principles and Normalization

Key Points – Design Principles for Relational Schemas

- The meaning of each relation should be easy to describe.
  - avoid combining multiple entity types and relationship types into one relation
    - tradeoff may be that the schema no longer captures the total participation constraint
- Reduce redundant information and avoid update anomalies.
- Reduce the number of NULL values.
  - avoid creating relations where attributes are often NULL
- Don't allow the possibility of generating spurious tuples.
  - occurs when a foreign key refers to something other than the primary key of the other table
  - can occur if tables are decomposed improperly

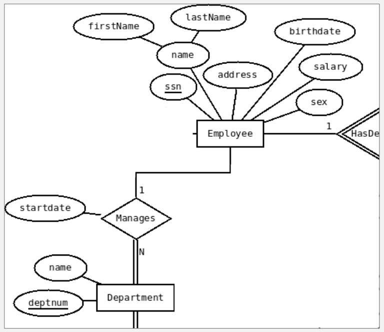
Update Anomalies

- *update anomalies* are problematic conditions that can arise during/from updates
- consider a merged relation approach for WORKS_IN




```
WORKS_IN(name,ssn,birthdate,
         address,deptnum,
         deptname)
```

  - an *insertion anomaly* occurs when insertion is blocked because required values are missing
    - e.g. can't add a department with no employees
  - a *deletion anomaly* occurs when deletion of some information results in the loss of something else, or some deletions need to be handled specially
    - e.g. deleting the last employee from a department means department info is lost
  - a *modification anomaly* occurs when information is duplicated and it is possible to update one copy without updating the others
    - e.g. changing part of a department's information requires changing every tuple where an employee works for that department



Using the merged relation approach for Manages results in a single relation for the Employee, Department, and Managers entity types. The following is one possibility:

```
Department(deptnum,deptname,ssn,empname,address,birthdate,salary,sex,st
artdate)
```

- Department.ssn NOT NULL

What kind(s) of update anomalies can occur?

insertion anomaly	4 respondents	80 %
deletion anomaly	4 respondents	80 %
modification anomaly	5 respondents	100 %
none of the above		0 %

Using the foreign key approach for Manages results in a single relation for Manages and either Employee or Department. The following is one possibility:

```
Employee(ssn,firstName,lastName,address,birthdate,salary,sex,deptnum,st
artdate)
Department(deptnum,name)
```

- Employee.deptnum -> Department.deptnum
- Employee.deptnum is NOT NULL

What kind(s) of update anomalies can occur?

insertion anomaly	5 respondents	100 %
deletion anomaly	2 respondents	40 %
modification anomaly	2 respondents	40 %
none of the above		0 %

- the existence of update anomalies means that these are not good mappings

Spurious Tuples

original data

ssn	projnum	hours	name	projname	projloc
123456789	1	32.5	John B Smith	ProductX	Bellaire
453453453	1	20.0	Joyce A English	ProductX	Houston

```
WORKS_ON(ssn,projnum,hours,name)
PROJECT(projnum,projname,projloc)
```

`WORKS_ON.projnum → PROJECT.projnum`

ssn	projnum	hours	name
123456789	1	32.5	John B Smith
453453453	1	20.0	Joyce A English

projnum	projname	projloc
1	ProductX	Bellaire
1	ProductX	Houston

combining the tables results in more tuples than we started with – and no way to reconstruct just the original

ssn	projnum	hours	name	projname	projloc
123456789	1	32.5	John B Smith	ProductX	Bellaire
123456789	1	32.5	John B Smith	ProductX	Houston
453453453	1	20.0	Joyce A English	ProductX	Bellaire
453453453	1	20.0	Joyce A English	ProductX	Houston

- getting spurious tuples means this is not a good decomposition

Spurious Tuples

- avoiding problems
  - model carefully and identify all functional dependencies
    - an employee works for a particular project in a particular place, not just a particular project
  - ensure that FKs only refer to complete PKs

original data

ssn	projnum	hours	name	projname	projloc
123456789	1	32.5	John B Smith	ProductX	Bellaire
453453453	1	20.0	Joyce A English	ProductX	Houston

```
WORKS_ON(ssn,projnum,hours,name,projloc)
PROJECT(projnum,projname,projloc)
```

`WORKS_ON.projnum,WORKS_ON.projloc → PROJECT.projnum,PROJECT.projloc`

Key Points – Normal Forms and Normalization

We want our relational schema to satisfy these design principles.

Careful design at the ER stage and careful application of the ER→relational model transformation rules can produce a good relational design.

- and problems detected can often be fixed up by hand

Normalization provides a formal method for removing redundancies.

- work through a series of *normal forms*, identifying functional dependencies that violate the definition of the normal form and splitting the relation to fix those problems
  - each normal form prohibits a certain kind of redundancy

Key Points – Normal Forms

- First Normal Form (1NF)
  - eliminates things that aren't valid relational schemas
- Second Normal Form (2NF)
  - eliminates dependencies on partial keys
- Third Normal Form (3NF)
  - eliminates dependencies on non-key attributes
- Boyce-Codd Normal Form (BCNF)
  - eliminates all redundancy due to functional dependencies
- Fourth Normal Form (4NF)
  - does not allow multiple multivalued attributes or M:N relationships in one relation
- Fifth Normal Form (5NF)
  - eliminates info that can be constructed from smaller pieces

Questions

Is there an ideal normal form every database design should achieve?

- achieving 3NF or BCNF is typically good enough
- higher normal forms are not as useful in practice
  - violations are relatively rare and can be hard to detect
- may choose not to normalize to the highest possible normal form
  - balance removal of redundancy (more relations) with performance (fewer relations reduces need for joins)

Key Points – Normal Forms Definitions

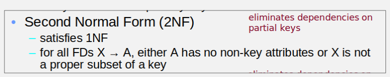
- First Normal Form (1NF) — eliminates things that aren't valid relational schemas
  - all values are atomic – no composite attributes
  - each tuple has a single value for each attribute (can be NULL) – no multivalued attributes
  - every relation has a primary key
- Second Normal Form (2NF) — eliminates dependencies on partial keys
  - satisfies 1NF
  - for all FDs X → A, either A has no non-key attributes or X is not a proper subset of a key
- Third Normal Form (3NF) — eliminates dependencies on non-key attributes
  - satisfies 2NF
  - for all non-trivial FDs X → A, either X is a superkey or everything in A-X is part of a key
- Boyce-Codd Normal Form (BCNF) — eliminates all redundancy due to functional dependencies
  - for all non-trivial FDs X → A, X is a superkey
    - in a *trivial FD* X → A, X is a superset of A
    - a *superkey* is a set of attributes which uniquely identify tuples in the relation

Questions

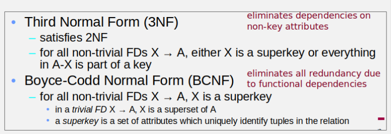
Is it always the case that a higher normal form implies a lower normal form?

- e.g. if a schema is in 2NF, will it always also be 1NF?

- normalization is meant to be progressive
  - move from lower to higher normal forms
- in many cases, satisfying a lower normal form is explicitly included in the definition of a higher normal form



- BCNF is a stronger version of 3NF – anything satisfying BCNF is automatically 3NF



Functional Dependencies

A *functional dependency* X → Y occurs when the value of a set of attributes X completely determines the value of another set of attributes Y.

- "there can be only one" – there can be only one instance of a particular set of values for Y for a particular set of values for X
  - does not mean that it is possible to compute the values for Y from the values of X
- existence of functional dependencies depends on the semantics of the attributes

Example.

STUDENT(<u>sid</u>,name,classyear,dean)

- classyear → dean [deans are assigned by class year]

Questions

What is a (proper) subset of a key?

- a *key* is a set of attributes which together uniquely identify tuples in the relation
- a *subset* of a set is made up of zero or more elements of the set
- a *proper subset* must have fewer elements than the set
  - a set S is a subset of itself
  - a set S is not a proper subset of itself

Consider the following relation:

```
REGISTERED(student,course,instructor,major)
```

The functional dependencies are:

- student → major
- course → instructor

What normal form(s) is this schema in? Choose all that apply.

1NF	4 respondents	**80 %**	✓
2NF	2 respondents	40 %	
3NF		0 %	
BCNF		0 %	
none of these		0 %	

Consider the following relations:

```
REGISTERED(student,course)
STUDENT(student,major)
COURSE(course,title)
```

The functional dependencies are:

- student → major
- course → ~~instructor~~ title

What normal form(s) is this schema in? Choose all that apply.

1NF	3 respondents	**60 %**	✓
2NF	3 respondents	**60 %**	✓
3NF	3 respondents	**60 %**	✓
BCNF	2 respondents	**40 %**	✓
none of these	1 respondent	20 %	

Consider the following relations:

```
LOTS(propertyid,county,lotnum,area,price,taxrate)
```

The functional dependencies are:

- county,lotnum → propertyid
- county → taxrate
- area → price

What normal form(s) is this schema in? Choose all that apply.

1NF	3 respondents	**60 %**	✓
2NF	1 respondent	**20 %**	✓
3NF		0 %	
BCNF	1 respondent	20 %	
none of these	1 respondent	20 %	

Normalization Process

Fix normal form violations by *decomposing* relations.

- for a functional dependency $X \rightarrow Y$, split R into relations XY and R-Y

When decomposing a relation R to satisfy a normal form, must ensure

- lossless join
  - cannot get spurious tuples

splitting R into XY and R-Y is lossless as long as $X \cap Y = \varnothing$ i.e. X and Y do not have any attributes in common

- dependency preservation
  - each functional dependency of R is represented in some relation or can be derived from those that are