DESIGN THEORY FOR RELATIONAL DATABASES

Introduction

- There are always many different schemas for a given set of data.
- E.g., you could combine or divide tables.
- How do you pick a schema? Which is better? What does "better" mean?
- Fortunately, there are some principles to guide us.

Schemas and Constraints

- Consider the following sets of schemas:

  Students(macid, name, email)

  vs.

  Students(macid, name)
  Emails(macid, address)

- Consider also:

  House(street, city, value, owner, propertyTax)

  vs.

  House(street, city, value, owner)

  TaxRates(city, value, propertyTax)

Constraints are domain-dependent

Acknowledgements: R. J. Miller, M. Papagelis

Avoid redundancy

This table has redundant data, and that can lead to anomalies.

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

- Update anomaly: if Janeway is transferred to *Intrepid*, will we remember to change each of her tuples?
- Deletion anomaly: If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.

Database Design Theory

- It allows us to improve a schema systematically.
- General idea:
  - Express constraints on the data
  - Use these to decompose the relations
- Ultimately, get a schema that is in a "normal form" that guarantees good properties, such as no anomalies.
- "Normal" in the sense of conforming to a standard.
- The process of converting a schema to a normal form is called normalization.

Part I:
Functional Dependency Theory

Keys

- *K* is a *key* for *R* if
  K uniquely determines all of *R*, and no proper subset of *K* does.
- *K* is a *superkey* for relation *R* if *K* contains a key for *R*.
  ("superkey" is short for "superset of key".)

Example

RegNum	Surname	FirstName	BirthDate	DegreeProg
284328	Smith	Luigi	29/04/59	Computing
296328	Smith	John	29/04/59	Computing
587614	Smith	Lucy	01/05/61	Engineering
934856	Black	Lucy	01/05/61	Fine Art
965536	Black	Lucy	05/03/58	Fine Art

- **`RegNum`** is a key: i.e., **`RegNum`** is a superkey and it contains a sole attribute, so it is minimal.
- **`{Surname, Firstname, BirthDate}`** is another key

Functional Dependencies

- Need a special type of constraint to help us with normalization
- $X \rightarrow Y$ is an assertion about a relation R that whenever two tuples of R agree on all the attributes in set X, they must also agree on all attributes in set Y.
- E.g., suppose X = {AB}, Y = {C}

R

A	B	C
x1	y1	c2
x1	y1	c2
x2	y2	c3
x2	y2	c3

Functional Dependencies

- Say "$X \rightarrow Y$ holds in R."
  "X functionally determines Y."
- Convention: …, X, Y, Z represent sets of attributes; A, B, C,… represent single attributes.
- Convention: no braces used for sets of attributes, just ABC, rather than $\{A,B,C\}$.

Why “functional dependency”?

- “dependency” because the value of Y depends on the value of X.
- “functional” because there is a mathematical function that takes a value for X and gives a *unique* value for Y.

Properties about FDs

- Rules
  - Splitting/combining
  - Trivial FDs
  - Armstrong's Axioms
- Algorithms related to FDs
  - the closure of a set of attributes of a relation
  - a minimal basis of a relation

Splitting Right Sides of FDs

- $X \rightarrow A_1A_2 \ldots A_n$ holds for R exactly when each of $X \rightarrow A_1$, $X \rightarrow A_2$,…, $X \rightarrow A_n$ hold for R.
- Example: $A \rightarrow BC$ is equivalent to $A \rightarrow B$ and $A \rightarrow C$.
- Combining: if $A \rightarrow F$ and $A \rightarrow G$, then $A \rightarrow FG$

- There is no splitting rule for the left side
  - $ABC \rightarrow DEF$ is NOT the same as $AB \rightarrow DEF$ and $C \rightarrow DEF$!
- We'll generally express FDs with singleton right sides.

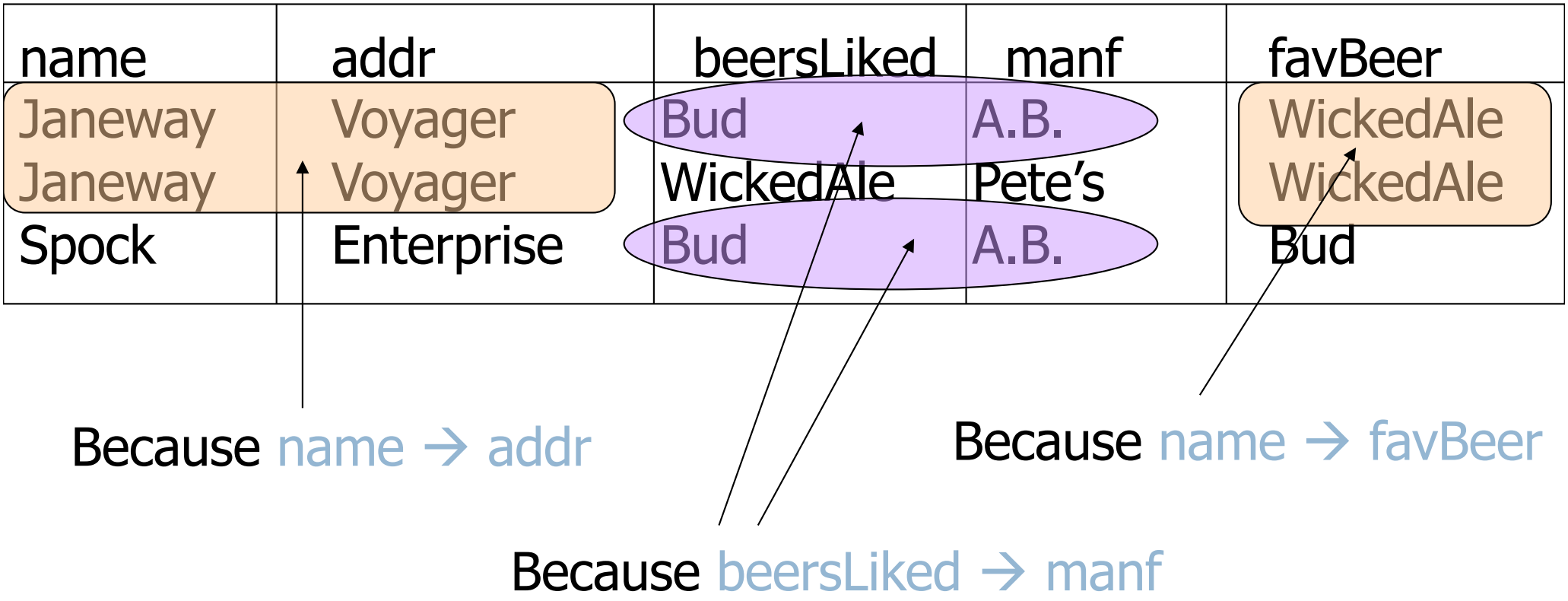
Example: FDs

Drinkers(name, addr, beersLiked, manf, favBeer)

Reasonable FDs to assert:

- name → addr, favBeer.
  - Note this FD is the same as: name → addr and name → favBeer.
- beersLiked → manf

Example: Possible Data



Trivial FDs

- Not all functional dependencies are useful
  - A → A always holds
  - ABC → A also always holds (right side is subset of left side)
- FD with an attribute on both sides
  - ABC → AD becomes ABC → D
  - Or, in singleton form, delete trivial FDs

    ABC → A and ABC → D becomes just ABC → D

Superkey

Drinkers(name, addr, beersLiked, manf, favBeer)

- {name, beersLiked} is a superkey because together these attributes determine all the other attributes.
  - name → addr, favBeer
  - beersLiked → manf

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

Example: Key

- {name, beersLiked} is a key because neither {name} nor {beersLiked} is a key on its own.
  - name doesn't → manf; beersLiked doesn't → addr.
- There are no other keys, but lots of superkeys.
  - Any superset of {name, beersLiked}.

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

FDs are a generalization of keys

- Functional dependency: X → Y
- Superkey: X → R
- A superkey must include all the attributes of the relation on the RHS.
- An FD can involve just a subset of them
  - Example:

Houses (street, city, value, owner, tax)

- street,city → value, owner, tax *(both FD and key)*
- city,value → tax *(FD only)*

Identifying functional dependencies

- FDs are domain knowledge
  - Intrinsic features of the data you're dealing with
  - Something you know (or assume) about the data
- Database engine cannot identify FDs for you
  - Designer must specify them as part of schema
  - DBMS can only enforce FDs when told to
- DBMS cannot "optimize" FDs either
  - It has only a finite sample of the data
  - An FD constrains the entire domain