EPBI 414

Unit 3
Relational Databases & Normalization

The Recap - Unit 2

- Complexities in storing numeric, character, categorical, and date data
- Basic data structures: variables, arrays, nested arrays and tables
- Tools for abstraction and data collection: case report forms

Unit 3 Overview

- The relational data model
  - Propositions and predicates
  - Defining relations (attributes and tuples)
  - Constraints and keys (primary and foreign)
  - Normalization and the single source of truth
  - Relational database management systems
- Documenting relational data structures: complex CRFs

Warning: logic ahead

The relational data structure is built on predicate logic and set theory.

This is a very general overview. Do not panic. You will not be required to derive any foundational principles!

The relational data model

- Published in 1970 by E.F. Codd
  - Further revised by Chris Date and Hugh Darwen
- Massive advancement in data storage
  - In practice, rapidly became dominant database system (until recently)
- Technically, no system purely adheres to the theoretical model
  - But we can make it work

Propositions

- A proposition is a statement which falls into either two categories: TRUE or FALSE

- A proposition:

  "Thomas Rehman, with ID 12345, is an adjunct instructor at CWRU."

Unknown vs. unknowable

- Propositions must ***always*** resolve to either true or false
- It does not matter if you don't ***know*** if it is true or false
- The statement must have only one value, true or false

What is a database?

"A database is a set of true propositions."
 - Hugh Darwen

Generalizing to predicates

- A ***predicate*** is the general form of a series of propositions
- In other words: if a series of statements have the same logical structure, the ***predicate*** is that underlying structure
- An example...

Example propositions

"Thomas Rehman, with ID 12345, is an adjunct instructor at CWRU."

"Mendel Singer, with ID 34567, is an associate professor at CWRU."

"Jane Smith, with ID 23456, is a graduate student at CWRU."

Where's the predicate?

"**Thomas Rehman**, with ID **12345**, is **an adjunct instructor** at CWRU."

"**Mendel Singer**, with ID **34567**, is **an associate professor** at CWRU."

"**Jane Smith**, with ID **23456**, is **a graduate student** at CWRU."

Predicate revealed

"**<NAME>**, with ID **<ID>**, is **<POSITION>** at CWRU."

Let us call this predicate C1

- We call the colored items ***parameters***
- Assigning a given set of values to the parameters creates a proposition
  - The resulting proposition will be true or false

The utility of predicates

- Predicates are tools which transform sets of values into propositions
  - One value for each parameter in the predicate
- The proposition can be true or false
- A database is a system which stores values that create true propositions
  - Why would you store false propositions?

Revisiting "What is a DB?"

"A database is a set of true propositions."
 - Hugh Darwen

From predicate to relation

- A predicate is a logical structure
- A ***relation*** stores both the structure and the sets of values which, when substituted into the predicate, create true propositions
- We are used to considering relations in table format

Predicate C1 as relation

NAME (character)	ID (integer)	POSITION (character)
Thomas Rehman	12345	an adjunct instructor
Mendel Singer	34567	an associate professor
Jane Smith	23456	a graduate student

Relation parts: header

NAME (character)	ID (integer)	POSITION (character)
Thomas Rehman	12345	an adjunct instructor
Mendel Singer	34567	an associate professor
Jane Smith	23456	a graduate student

Relation parts: body

NAME (character)	ID (integer)	POSITION (character)
Thomas Rehman	12345	an adjunct instructor
Mendel Singer	34567	an associate professor
Jane Smith	23456	a graduate student

The header: structure

- The header consists of attributes
- Each has a ***name*** and a ***data type***
  - Some also define the ***data domain*** in the header; this is a little more nuanced
  - More about ***data domain*** and ***data type*** later
- Header defines what tuples are part of this relation

Tuples?

- A ***tuple*** is strictly defined as a:
  - finite
  - ordered
  - list of elements
- For a relational database, replace "ordered" with "named"

Relation parts: a tuple

NAME (character)	ID (integer)	POSITION (character)
Thomas Rehman	12345	an adjunct instructor
Mendel Singer	34567	an associate professor
Jane Smith	23456	a graduate student

The body: values

- The body is the set of tuples that meet the following criteria:
  - Each tuple must have exactly one element for each ***attribute*** in the header, where that element's value is the correct type
  - Each tuple must produce a true proposition when the values of the tuple are substituted into the matching parameters of the predicate that defines the relation
    - "A database is a set of true propositions"

Data types and domains

- You may recall this from a previous class:

  "Type is fundamental to a piece of data. Range checks are applied on top of type."

- In the relational model, both ***data type*** and ***data domain*** can be expressed as sets
  - However, the type is generally more fundamental than the domain

Data types as sets

- A ***data type*** is still a fundamental aspect of data
  - For instance, an ***integer*** data type could be expressed as the set of all integers, $\mathbb{Z}$
  - A ***decimal*** data type could be expressed as:

    $D = \{x \mid x = a / 1000, a \in \mathbb{Z}\}$
  - A ***character*** data type could be expressed as:

    $C = \{A, B, C, D, \ldots x, y, z\}$

Data domains as sets

- A ***data domain*** generally is applied after the ***data type***
  - For instance, storing gender as M, F, or U means using the set `{M, F, U}`, all of which are character
  - Storing values between 0 and 30 means using the set `{0 ... 30}` on a numeric variable
- One could reasonably ask: why not simply use the ***data domain*** as the ***data type***?

The main difference

- A ***data type*** both tells you the valid values, and expresses the ***operators*** that can work on that data
  - For instance, numbers can be added; words cannot
- A ***data domain*** enumerates what the acceptable values are, but makes no comments about operators

Theory vs. practice

- In theory, you could define custom ***data types*** for each of your attributes
  - This could let you build ranges into your data types
  - It would also require you to define operators and such for each
- In practice, you will use the built-in types in your database software, and then apply constraints

Expressing data domains

- The power of the relational model is that it permits you to apply various types of ***constraints*** to your data

- These ***constraints*** prevent you from entering inconsistent data

- They are stored in database system itself, so they are part of your data's structure

Types of constraint

- There are a variety of constraint types, and they often vary by system

- Some fairly common ones:
  - UNIQUE
  - CHECK
  - PRIMARY KEY
  - FOREIGN KEY
  - NOT NULL

Expressing constraints

- Theoretically, all constraints can be expressed using set comparisons, specifically, subset
  - Just like data types
- However, most systems use more traditional logical / mathematical terms
  - These can always be reduced to some type of set
  - Possibly an infinite set

Important constraints, pt 1

- **NOT NULL**: The columns do not contain null or empty values
- **UNIQUE**: The columns contain unique values for every row in that table
- **PRIMARY KEY**: UNIQUE + NOT NULL
  - Uniquely identifies each row in the table

Important constraints, pt 2

- **FOREIGN KEY**: The values in the column are constrained by a column in another table
  - This is ***almost always*** the primary key of that table
  - You can consider it to always be the case - exceptions are unusual
- **CHECK**: The values are constrained by some logical check performed by the system
  - The check is stored as an aspect of the column

Examples of constraints

- A race that must be White, Black, or Asian:
  - `race ∈ {White,Black,Asian}`
- An age which must be an integer, greater than 18:
  - `age ∈ {18,19,20,...}`
- Note that you could implement these in different ways in each database

Connecting tables

- One of the most powerful uses of a constraint is to connect together two tables

- This is done using ***primary keys*** and ***foreign keys***
  - A primary key uniquely identifies each row

  - A foreign key is constrained by the values in another table's primary key

The primary key

- Every table ***should*** have a primary key
  - Strictly speaking, the primary key is required
  - Most database software will allow you to not declare one (this is bad)
- The primary key serves to uniquely identify each tuple in the relation
  - Again, some database software will allow you to have duplicate rows - you should avoid this

Picking primary keys

- Sometimes, the primary key is obvious
  - In a table of patients, the patient ID
  - In a table of colors, the color names or hex code
- Other times, you might need to declare or create a primary key
  - In a table of medications, you might need to declare a primary key (like med_id)
  - This is often done with auto-incrementing numbers

Create vs. combine?

- Sometimes, you might use multiple columns as the primary key
- Other times, you might create a column to do this
- Depends on context and what you are trying to do

Foreign keys

- A ***foreign key*** is a column in one table that is constrained by a column in another table
  - This is almost always the ***primary key*** of another table
- In other words, the value of a foreign key is an element of a column in another table

```
forn.key ∈ {other_table_key}
```

The magic of foreign keys

- A foreign key ensures that you cannot have invalid values in one table, ***based on another table***

- It allows you to build tables that are dependant on other tables
  - Permits much more complicated representations of data
  - Can turn constraints into other tables in database

An example of keys

Table: cwru_personnel

pers_id	staff_name	staff_position
12345	Thomas Rehman	Adjunct Instructor
34567	Mendel Singer	Associate Professor
23456	Jane Smith	Graduate Student

Table: cwru_courses

class_id	instructor_id	class_name
101	12345	EPBI414
102	34567	MPHP405

The primary keys

Table: cwru_personnel

pers_id	staff_name	staff_position
12345	Thomas Rehman	Adjunct Instructor
34567	Mendel Singer	Associate Professor
23456	Jane Smith	Graduate Student

Table: cwru_courses

class_id	instructor_id	class_name
101	12345	EPBI414
102	34567	MPHP405

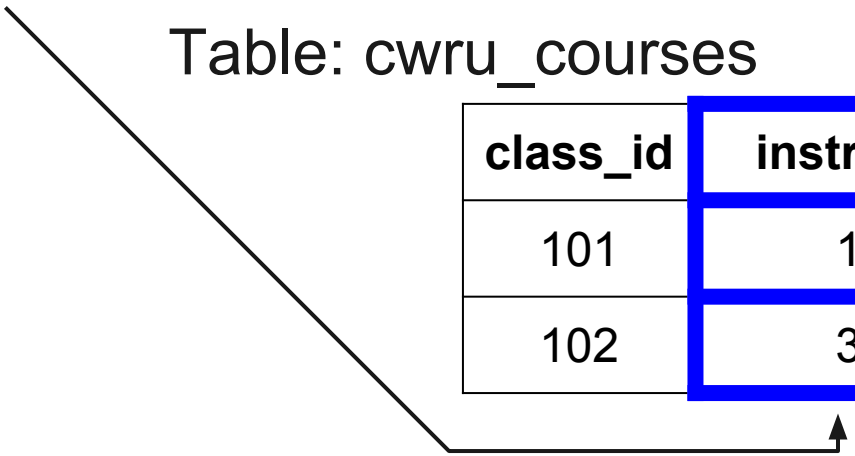
A foreign key

Table: cwru_personnel

pers_id	staff_name	staff_position
12345	Thomas Rehman	Adjunct Instructor
34567	Mendel Singer	Associate Professor
23456	Jane Smith	Graduate Student

Table: cwru_courses

class_id	instructor_id	class_name
101	12345	EPBI414
102	34567	MPHP405



In symbolic logic...

- The value in cwru_courses.instructor_id must be in the set of cwru_personnel.pers_id
  - In other words, every course must have an instructor found in the cwru_personnel table
- Symbolic representation:

$$\texttt{cwru\_courses.instructor\_id} \in \texttt{cwru\_pesonnel.pers\_id}$$

Types of relationship

- A ***one-to-one*** relationship connects a single row in table A to a single row in table B
- A ***one-to-many*** relationship connects a single row in table A to many rows in table B
- A ***many-to-many*** relationship connects many rows in table A to many rows in table B

Examples: one-to-one

- A one-to-one relationship is basically a row
- Often, we may not see much benefit in putting these in separate tables
- One key benefit can be performance
  - Sometimes, things that are not used frequently can be put in a different table

Examples: one-to-many

- Doctors and their patients
  - Each doctor is linked to many patients
- Professors and course leaders
  - Each professor may lead multiple courses
- Rooms to patients
  - Each room is assigned to one or more patients

Examples: many-to-many

- Diseases and their symptoms
  - Many diseases may share symptoms (i.e. multiple symptoms are connected to multiple diseases)
- Drugs and their side effects
  - Many drugs may share side effects, and many side effects may be caused by different drugs
- Professors and courses
  - Each course may have multiple professors; each professor may have multiple courses

Foreign as primary

- It is unusual to have a table where the primary key is also a foreign key
  - This is basically a one-to-one relationship
- However, it is fine to have a table where the primary key is comprised of multiple foreign keys
  - This is common for a many-to-many relationship

The single source of truth

- One powerful feature of constraints and keys is preserving a ***single source of truth*** (SSOT)
- Simply stated, this is the principle of ***only storing data*** once in the database
- This prevents inconsistency in the database

Without the SSOT

Table: cwru_personnel

pers_id	staff_name	staff_position
12345	Thomas Rehman	Adjunct Instructor
34567	Mendel Singer	Associate Professor
23456	Jane Smith	Graduate Student

Table: cwru_courses

class_id	instructor	class_name
101	Thomes Rahman	EPBI414
102	Mandil Singer	MPHP405

Database inconsistency

Table: cwru_personnel

pers_id	staff_name	staff_position
12345	Thomas Rehman	Adjunct Instructor
34567	Mendel Singer	Associate Professor
23456	Jane Smith	Graduate Student

Table: cwru_courses

class_id	instructor	class_name
101	Thomes Rahman	EPBI414
102	Mandil Singer	MPHP405

Break Time

Database normalization

- Builds on the concept of SSOT by recognizing whether a database is designed to ***prevent inconsistency***
- Databases can be classified into one of ***five normal forms*** based on their structure
  - Represented from 1NF to 5NF; we will discuss 1 - 3
- Higher NF = more consistent data

The 1st Normal Form (1NF)

- All records must contain the same number of fields
- Simplified: all rows of the table must have the same number of columns
- Alternative: all elements of a row must contain one datum

1NF - Example

Does not meet the first normal form:

pers_id	staff_name	email
12345	Thomas Rehman	tar9@case.edu tom@fake.net
34567	Mendel Singer	mes12@case.edu
23456	Jane Smith	not.real@fake.net

1NF - Corrected Example

Does meet the first normal form:

pers_id	staff_name	email
12345	Thomas Rehman	tar9@case.edu
12345	Thomas Rehman	tom@fake.net
34567	Mendel Singer	mes12@case.edu
23456	Jane Smith	not.real@fake.net

The 2nd Normal Form (2NF)

- To be 2NF, a table:
  - Must be 1NF
  - Must not contain any non-key fields that are dependent only upon a portion of the primary key
- In other words: no non-key field may depend on only a part of the primary key
  - This is only relevant when the primary key is more than one field

2NF - Example

Does not meet the second normal form:

personnel_id	degree_earned	degree_yr	office_loc
12345	BA	2007	Madison, WI
12345	MPH	2009	Madison, WI
34567	BA	1981	Cleveland, OH
34567	MS	1984	Cleveland, OH
34567	PhD	1991	Cleveland, OH

2NF - Corrected Example

Does meet the 2NF:

personnel_id	degree_earned	degree_yr
12345	BA	2007
12345	MPH	2009
34567	BA	1981
34567	MS	1984
34567	PhD	1991

personnel_id	office_loc
12345	Madison, WI
34567	Cleveland, OH

The 3rd Normal Form (3NF)

- To be 3NF, a table:
  - Must meet the 2NF
  - Must not contain any non-key columns which are ***not*** facts about a non-key column
- Simplified: a table must contain facts only about the variable(s) on which it is keyed
  - All non-key columns must be about all the keyed fields

3NF - Example

Does not meet the third normal form:

award	year	awardee_id	awardee_favorite_lang
Best Teacher	2009	123	R
Best Student	2009	456	Python
Best Teacher	2008	123	R
Best Student	2008	789	Java

3NF - Corrected Example

Does meet the third normal form:

award	year	awardee_id
Best Teacher	2009	123
Best Student	2009	456
Best Teacher	2008	123
Best Student	2008	789

personnel_id	favorite_ language
123	R
456	Python
789	Java

The three normal forms

"The data depends on the key (1NF), only the key (2NF), and nothing but the key (3NF), so help me Codd."

Higher normal forms

- The first three NFs came from Codd originally
- There are others (4NF, 5NF, BCNF)
  - Generally, databases can be called "normalized" if they meet 3NF
- Your need: understand normalization and how multidimensional data is stored

The RDBMS

- A ***R***elational ***D***ata***B***ase ***M***anagement ***S***ystem is what we would call "database software"

- Some common RDBMSs:
  - MySQL
  - PostgreSQL
  - SQLite
  - Microsoft SQL Server
  - Oracle Database

The ugly truth

Almost no "relational database management system" actually satisfies the relational model in full.

The convenient truth

Most of the time, this does not matter to what you will do as an analyst.

Flavors of RDBMS

- The main thing to know: there are minor (but important) differences between RDBMSs
- They often implement slightly different flavors of SQL (Structured Query Language)
- There are trade-offs and benefits to each package

Relational CRFs

- Many studies are more complex than a single table
- A complex study can often benefit from an RDBMS
- Documenting CRFs that use relational structures can be complex

When to go relational?

- When you need to collect "choose all that apply"
  - Though you can do this with binary variables
- Big case: when you need to record data but you don't know how much
  - i.e. When the question you ask may have one, two, or fifty answers

Example: racial categories

- In the second unit, we discussed race categories as binary variables
- An alternative is to use a relational data structure
- In that structure, the race of a subject would be stored in a separate table

Racial data in one table

pat_id	pat_name	pat_dob	race.wht	race.blk	race.ami	race.asn	race.pac
101	David Johnson	5/6/1943	1	0	1	0	0
102	Michael Risondo	8/12/1977	1	0	0	0	0
103	Robert Robs	11/16/2001	0	1	0	1	0

Race data in two tables

Table: patient_ids

pat_id	pat_name	pat_dob
101	David Johnson	5/6/1943
102	Michael Risondo	8/12/1977
103	Robert Robs	11/16/2001

Table: patient_races

pat_id	pat_race
101	White or Caucasian
101	American Indian or Alaska Native
102	White or Caucasian
103	Black or African American
103	Asian

Another example: meds

- A very common request: "Please record all the medications the patient is taking."
  - This is generally a disaster for most studies
  - Storing and analyzing this data can be hard
- Using an RDBMS makes this much easier
  - Also, allows for individuals to list variable amounts of data

An example meds table

Table: patient_ids

pat_id	pat_name	pat_dob
101	David Johnson	5/6/1943
102	Michael Risondo	8/12/1977
103	Robert Robs	11/16/2001

Table: patient_meds

pat_id	pat_med
101	alprazolam
101	adriamycin
102	aspirin
103	codeine
103	pyrimethamine

Annotating relations

- CRFs need annotated even if they "speak to" multiple tables
- One way to do this is to denote those relations with a dot (.)
  - For instance, [table1].variable1 indicates variable1 in table1
- Often, data from a CRF goes to multiple places

Complex CRF guide

- When making annotations, always indicate the table
  - Sometimes, you might omit this if the form has a designated table - then, just mark the non-standard fields
- On the data dictionary, use different tabs for different tables
  - You can use a placeholder row to indicate that a specific question is on another table

Complex CRF guide, cont.

- When designing, consider the normal forms and the SSOT
  - Don't collect or store data more than once
  - If you do - design a check for it
- Hire and partner with a good DBA or data developer
  - You will hate your life less if you do this!