Biological Database Design Week 5

Winter '07

Melanie Nelson, Ph.D.

Physical Database Design

- Physical design is process by which the logical design shown in the entity-relationship diagram is transformed into tables and constraints in the database
- Also decide storage details and indexing requirements
- Highly dependent on choice of RDBMS and usage requirements of the database
  - Therefore, I am not going to cover this in detail
  - For most small, research-type databases, physical design decisions will not be crucial
  - For large, production level databases, either take some more DB classes or hire a DBA!

Transforming ER Diagrams into Tables: General Rules

- Each entity is a table
- Each attribute is a column
- Unless an attribute is marked as optional (O), the columns has a NOT NULL constraint
- If the primary key is a system-generated unique identifier, it is a good idea to add UNIQUE constraints to columns that are part of an alternate key
  - This ensures that the actual data in the table is not duplicated
  - Remember to put constraint on combination of columns that make up the key (not on each column individually)

Transforming ER Diagrams into Tables: Exceptions

- May decide to store all subtypes in one table
  - Reduces number of joins needed
  - Downside: any attributes unique to one or the other subtypes must be NULLable for all subtypes
- May generalize: i.e., group some related entities into one table
  - I usually only do this if I can include it in my logical model, i.e., if the entities I'm grouping can be replaced by a supertype entity
  - Examples of this will be shown later in lecture

Transforming ER Diagrams into Tables: Exceptions

- May decide to remove some of the NOT NULL constraints
  - Usually done for ease of data loading
  - I don't recommend this: either the data is required or it isn't!
  - Better idea: temporarily disable a constraint for a batch load, and then reapply and flag (and deal with!) exceptions

Transforming ER Diagrams into Tables: Exceptions

- May decide not to apply UNIQUE constraints across large or complicated alternate keys
  - These constraints may unacceptably slow down data entry
  - May make a business decision not to look for duplicates (for instance, not looking for exact duplicate sequences from different sources)
  - Be careful: this creates a risk that data in table will be duplicated even though the (system-generated) primary key is unique!
  - Consider running a “data cleaning” program periodically

Denormalizing

- IF any denormalizing is done, it is done during physical design
- **I DON'T RECOMMEND DOING THIS!**
- Before denormalizing:
  - Tune queries
  - Add or tune indexes
  - Make sure developers are using bind variables (so that queries are cached)
  - Increase cache size
  - Understand the cost: perhaps users can accept slower queries when they realize that the alternative is to risk data integrity
  - Hire a good DBA, who can do all of the above better
- If you must denormalize, I recommend the use of "summary tables" or materialized views. More on this later in the lecture.

Biological Databases are Like Other Databases

- In many ways, biological databases are no different from other databases
  - Should follow good design practices that have been developed in ~30 years of work on relational databases
  - Should take advantage of the many excellent general database design and performance tuning resources that are available

Unusual Aspects of Biological Databases

- Large subject area, with many interrelationships among data
- Complex, constantly evolving “business rules”
- Special requirements of scientific culture
- Prevalence of complex data types and reliance on flat file formats

Handling the "Largeness" of Biology

- Biology encompasses many different "levels" of enquiry
  - Evolutionary and populations biology
  - Medicine and gross anatomy
  - Cell biology
  - Molecular biology and biochemistry

} An incomplete list!

- Biology is integrative
  - It is common to use information from many disciplines
  - Biological databases are beginning to cross disciplines, too

Handling the "Largeness" of Biology

- It is not possible to design a database that can handle all of biology
  - At least not in one iteration!
- Carefully define the scope of the database
  - Areas that are in scope should be addressed in full detail
  - Areas outside of scope can be simplified, or handled as text fields
- If scope still seems large, consider addressing only a portion of it in initial release

Handling "Largeness" of Biology

- A common mistake of DB designers with no biological background is to fail to stop detailed design at appropriate place
  - Will ask biologists to describe data, and biologists will oblige… but the full structure of that data might be out of scope
  - In general, DB designers with a background in biology are more likely to know when to stop describing the data in detail… but they should still define their DB scope!

Examples of Scope Boundaries

- The Protein Databank (PDB)
  - Database to store information on 3D structure of proteins and nucleic acids
  - In scope:
    - Coordinates of structures
    - Experimental details of how structures were determined
    - Information about the construct used to produce material used in experiments
  - Out of scope:
    - Regulation of gene that produces protein/nucleic acid *in vivo*
    - Audit trail of experiments used to determine structures
  - On the boundary:
    - Information about function of protein/nucleic acid in structure
    - Details about how any post-translational modifications on the protein were produced

Example of Scope Boundaries

- Database tracking interactions between enzymes and inhibitors
  - In scope
    - Exact sequences against which inhibition is measured
    - Relationships among inhibitors
    - Inhibition constants
  - Out of scope
    - Methods for synthesizing of inhibitors
    - Evolutionary relationships among homologous enzymes
  - On the boundary
    - Experimental protocol used to measure inhibition
    - Relationships between orthologous enzymes

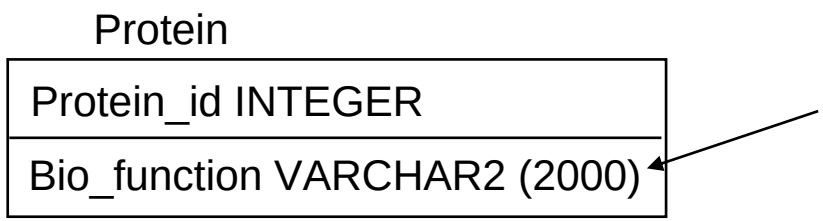
Scope Boundaries in DB Desgin

- Exact location of boundary is a property of the individual project
  - Different databases to store protein – inhibitor relationships may have different boundaries
    - One may want to track at the level of atomic interactions between protein and inhibitor
    - Another might not need atomic detail in interactions, but require more detailed information about experimental protocols
    - A third might need both!
  - Scope may be refined during detailed requirements gathering, but always know when you are redefining it!
- Scope creep (a close relative of feature creep) is a project killer

Scope Boundaries in DB Design

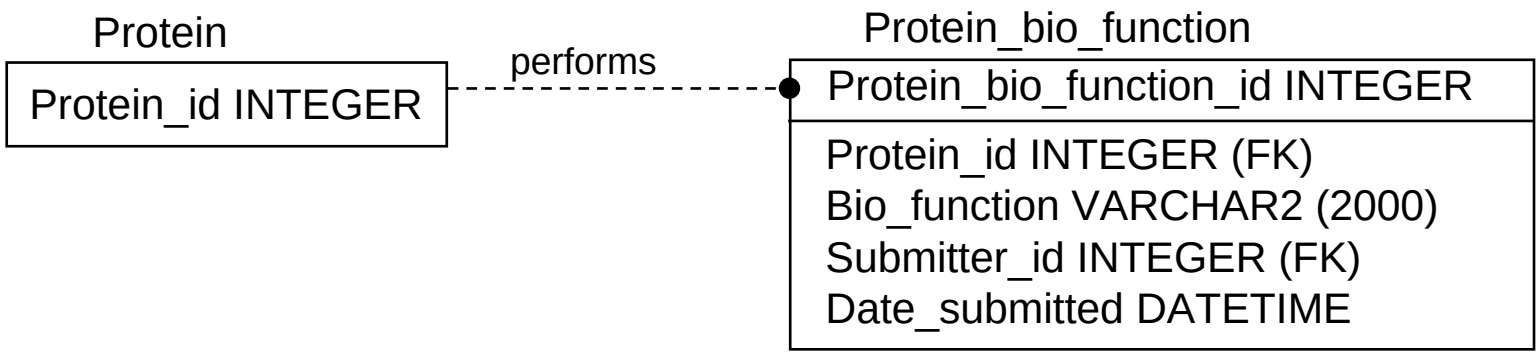
- Things that are out of scope may be modeled as free text (comment fields) or ignored
- Things on the boundary often need user-extensible classifications
  - These may also show up as free text
  - If an existing controlled vocabulary exists, using it gives you extra information “for free”
- Everything in scope must be modeled in full detail
  - Should incorporate existing controlled vocabularies where possible
  - May find you need to extend/modify existing vocabularies

Protein Function Out of Scope

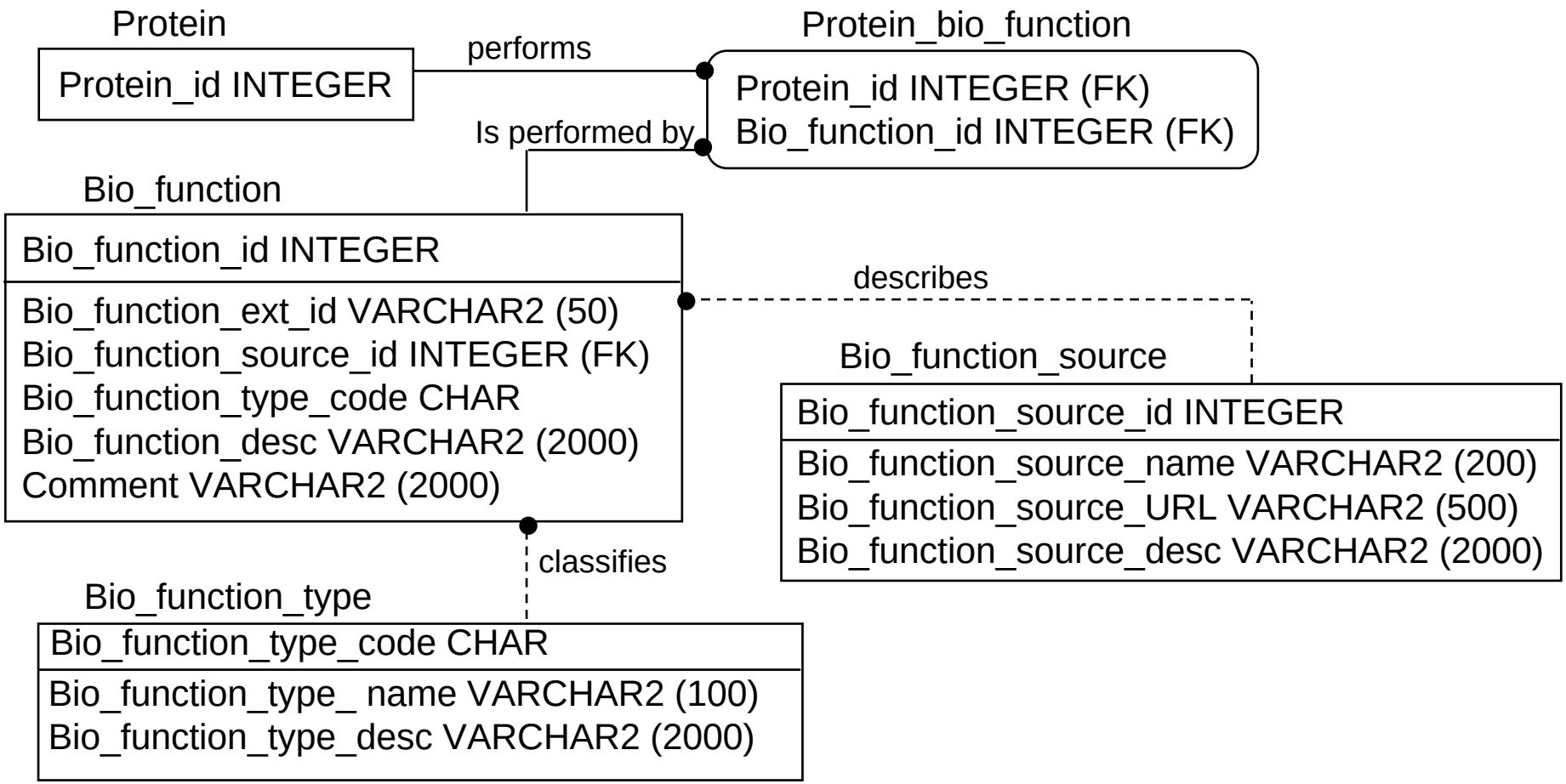


The biological function of a protein is stored as free text. This isn't much use for searching, but does provide context to scientists using the database. Multiple functions are all just listed in the free text field.

Even if biological function is outside of your scope, you may want to separate multiple functions, and track who provided each function:

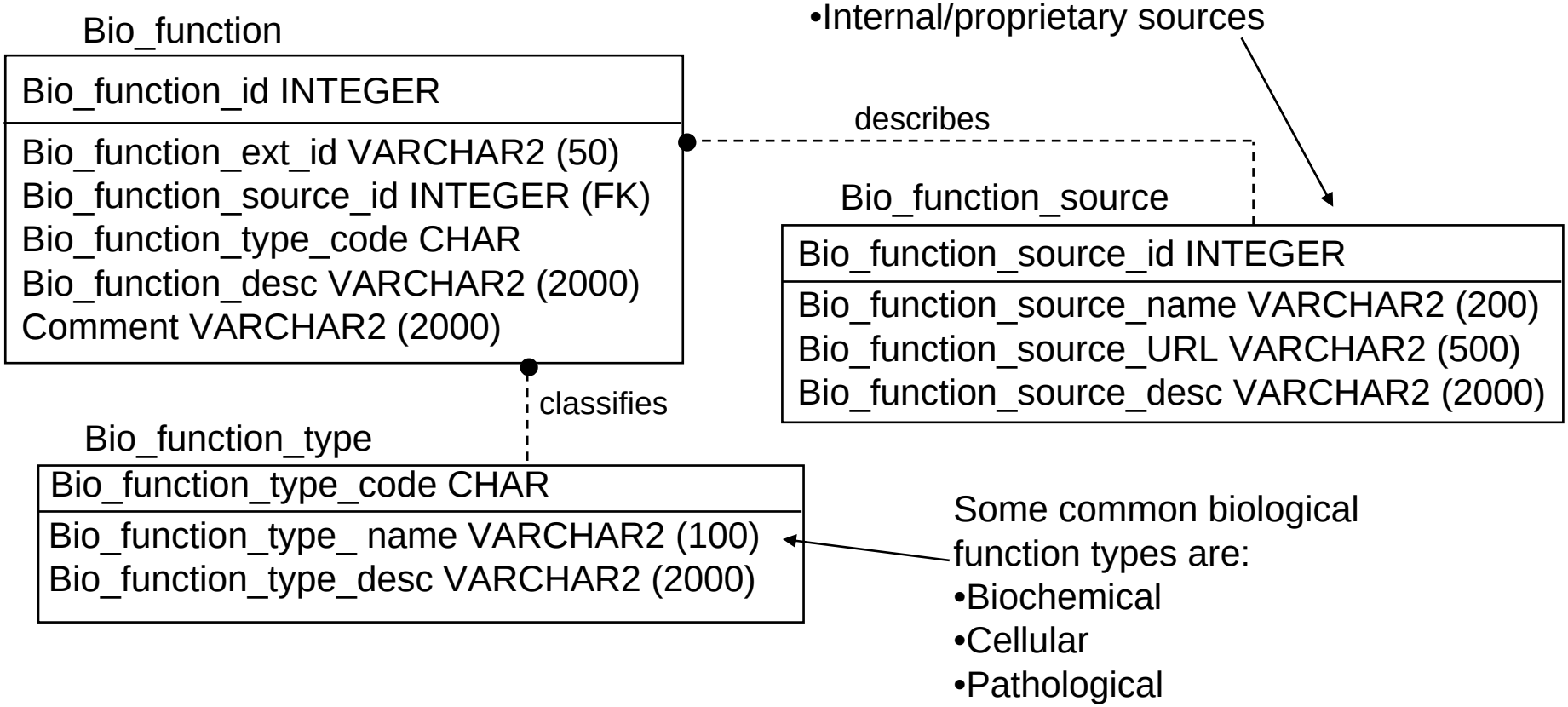


Protein Function on the Scope Boundary



Protein Function on the Scope Boundary

Sources of biological function include:
- Enzyme commission (EC) numbers
- Gene ontology (GO)
- Internal/proprietary sources



Some common biological function types are:
- Biochemical
- Cellular
- Pathological

Protein Function In Scope

- Handling protein function when it is within the scope of your database is quite complex and will almost certainly require generalization
  - If you try to model each type of function and each aspect of function separately, your data model will be very large
  - Non-generalized data model is also unlikely to be able to handle evolving field
- Some things to think about
  - Are all types of function within scope, or only one type (such as biochemical)?
  - Will you generate your own classification scheme and cross-reference it to public schemes like EC numbers or GO, or limit yourself to the public schemes
    - Scientists must be involved in this decision
    - Public schemes are inadequate for some applications
  - Where will the functional data come from?
    - You will almost certainly need to track source
- Example data model: the aMAZE database
  - www.amaze.ulb.ac.be
  - Representing and analysing molecular and cellular function using the computer. J. van Helden, et. al. (2000) Biol. Chem. 381:921-935

Handling Evolving "Business Rules"

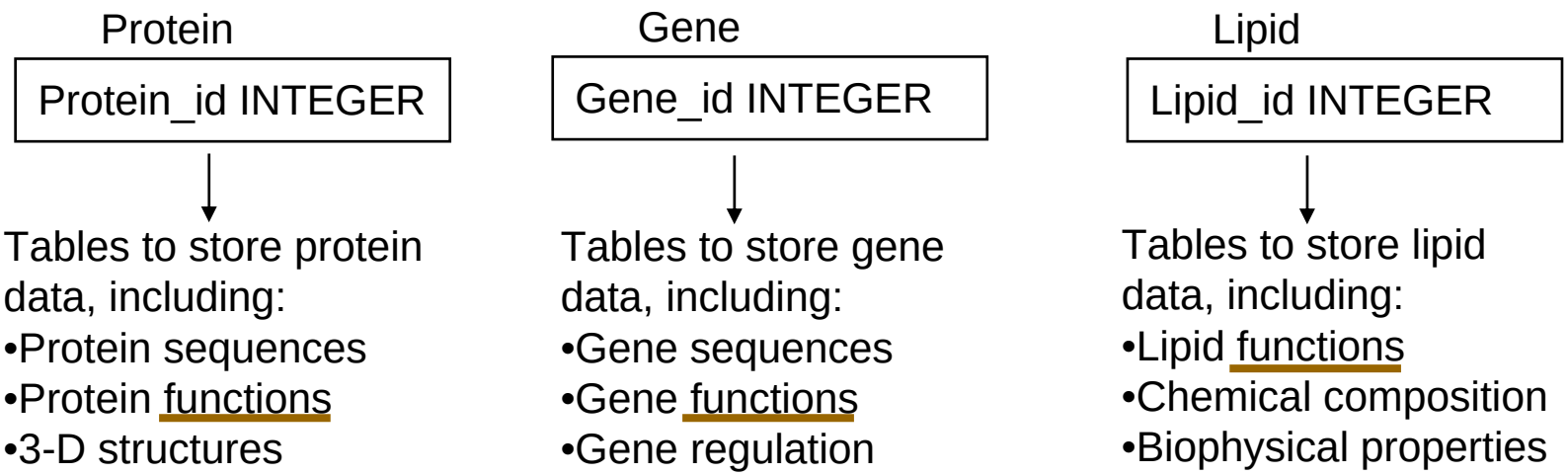
- Our understanding of biology is far from complete, and constantly changing
- Can never fully know the "business rules" for a biological database
  - The rules that are derived from biology are necessarily subject to change
  - Be wary of organization specific limitations on complexity when it is within scope
    - These are likely to change as the needs of the organization evolve

Use of Generalization

- Generalization in databases
  - Storing multiple subtypes of data in a table (or set of tables) that represent the supertype
  - May lead to some NULLable attributes
    - NULLs must be allowed on attributes that apply only to some of the subtypes
    - For this reason, may not be appropriate for some data
  - Can avoid NULLs by having separate subtype tables as well as the supertype table
- If you represent the supertype rather than the individual subtypes, your database schema is more robust to changes in data being stored
- Generalization can also decrease the size of your schema
  - However, abstraction may make schema more difficult to understand

Example of Generalization

- Database to store information about biological molecules
  - In requirements, scientists indicate they need to store data about proteins, genes, and lipids

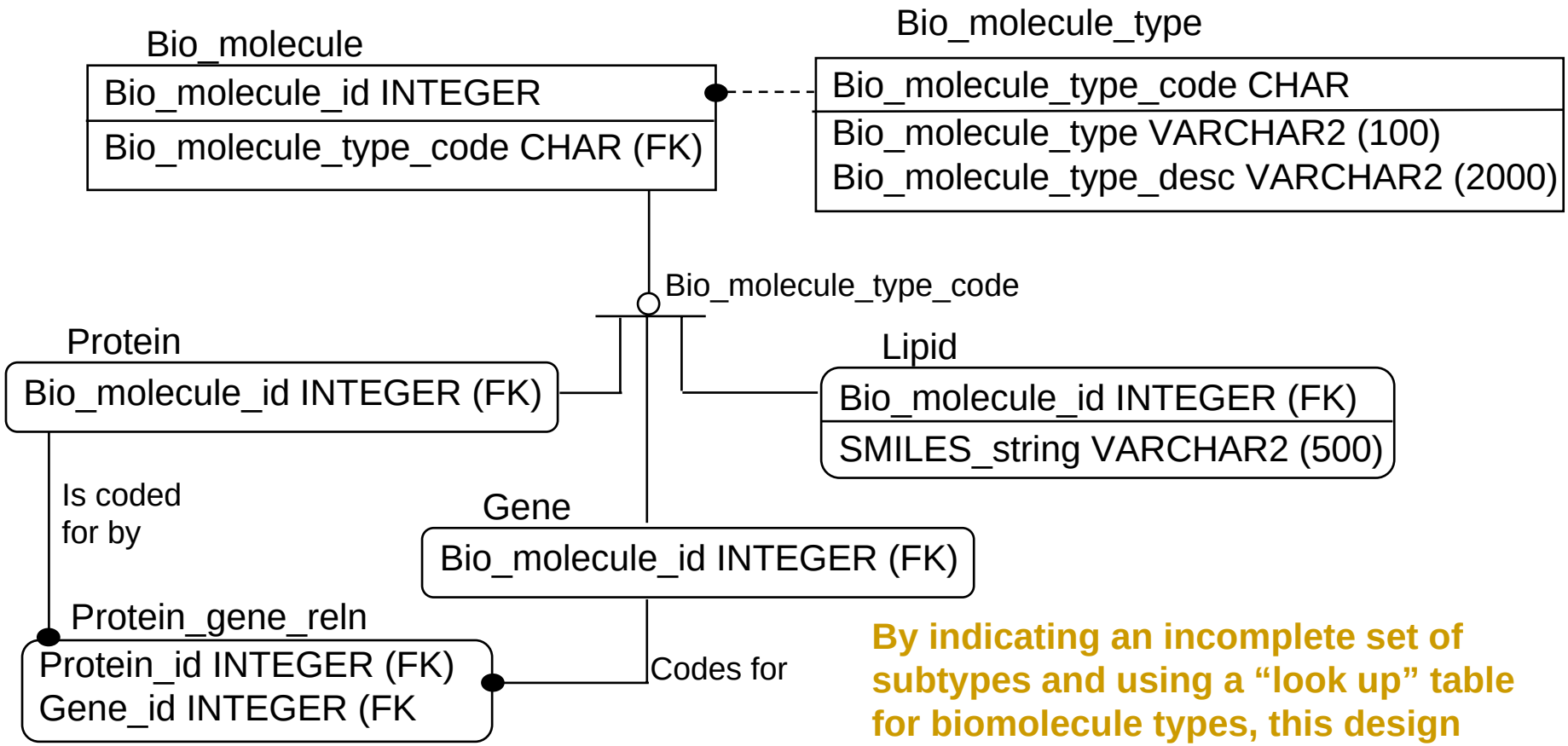


Similar or overlapping attribute lists are a clue that you should consider generalizing

Generalizing Biological Molecules

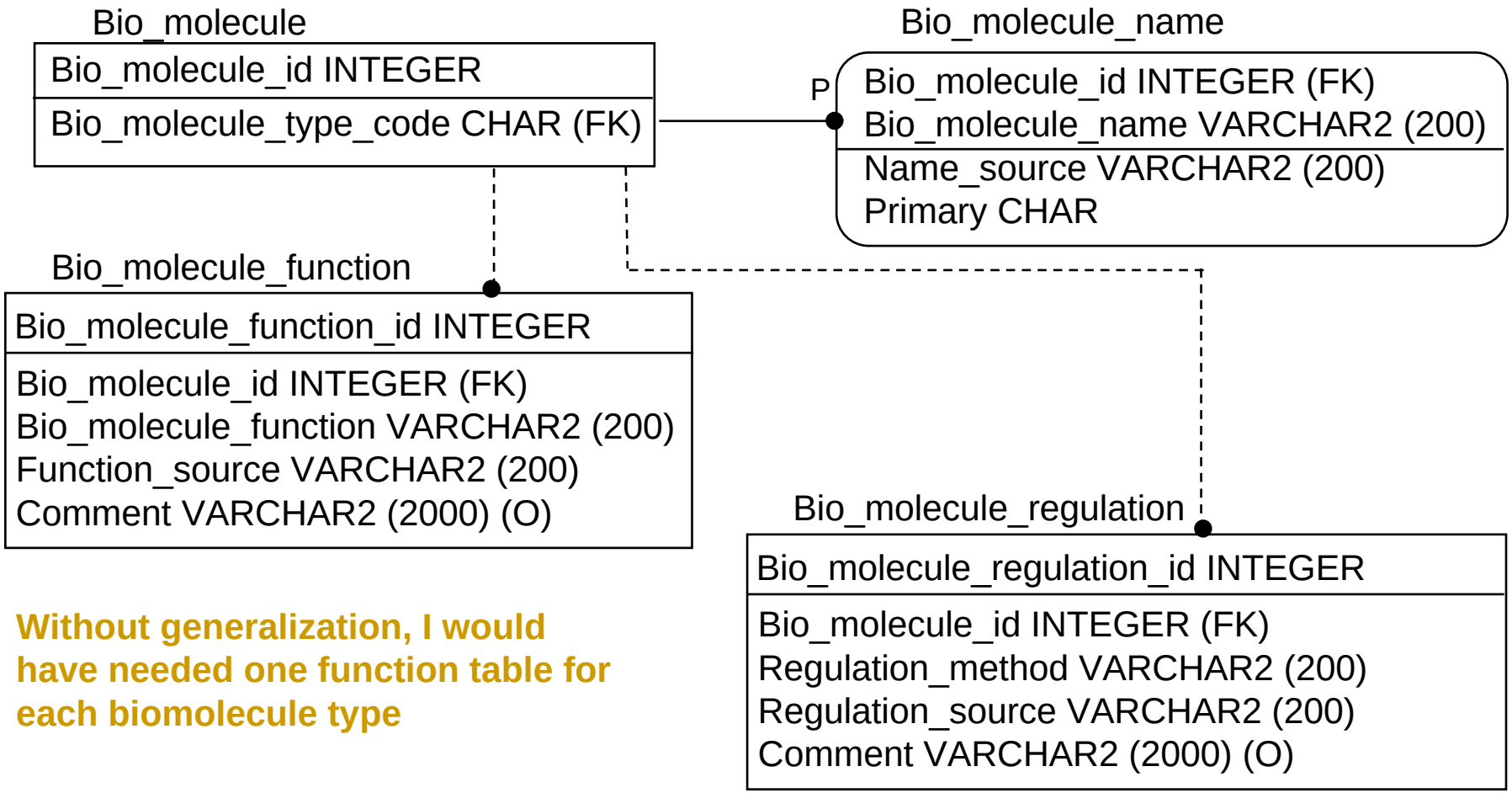
- Further consideration reveals that there are many types of biological molecules not covered in current requirements
  - mRNA
  - Ribosomal RNA
  - Small molecule metabolites
  - Inorganic ions
  - Etc.
- Scientists may not need to store information about these now, but this may change
- You can design your database so that it can store at least basic information about the other types of molecules
  - If (when!) scientists need to store information about these types, the DB can accommodate
  - May need to add subtype tables to handle data specific to the new types

Generalizing Biological Molecules



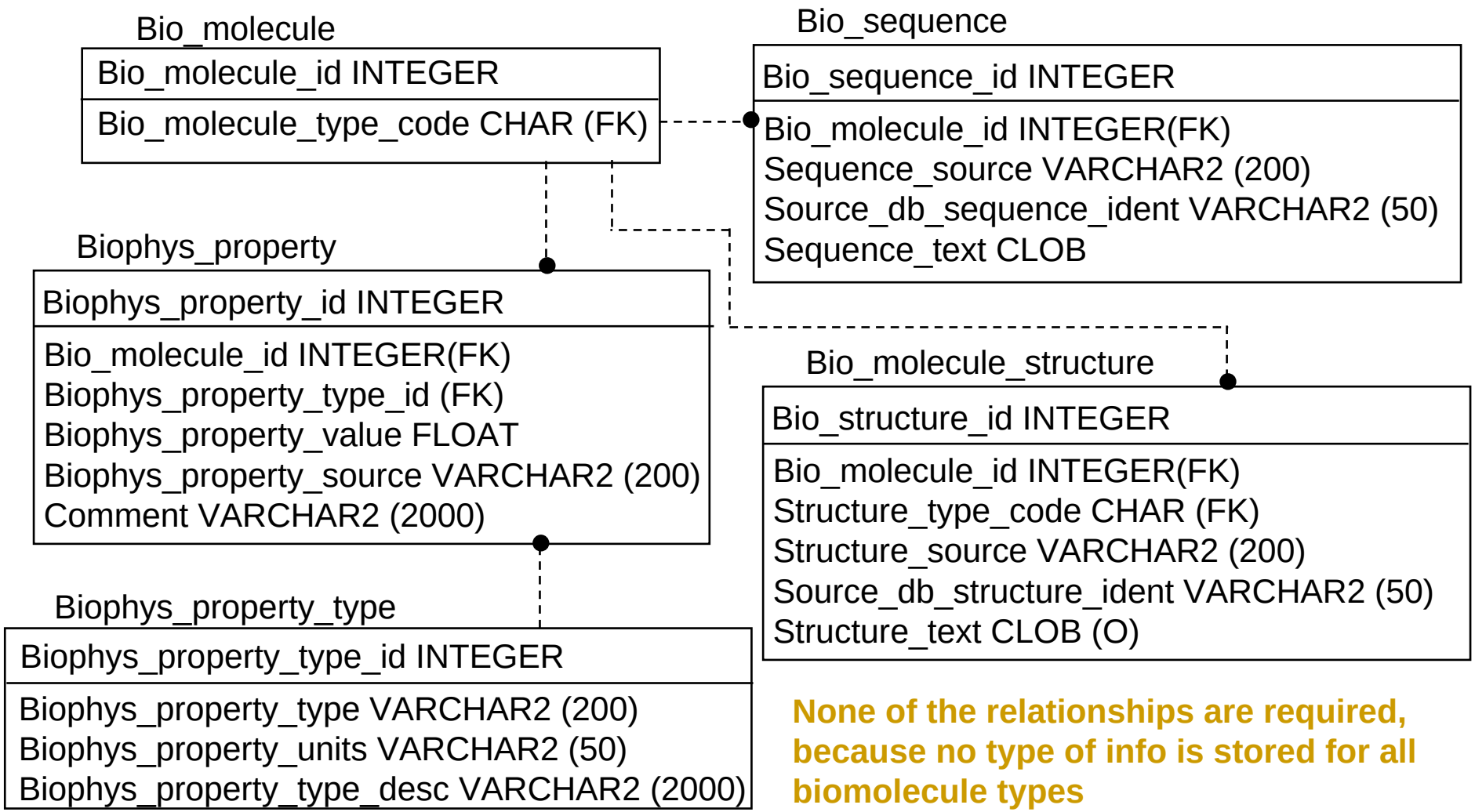
By indicating an incomplete set of subtypes and using a "look up" table for biomolecule types, this design allows basic information to be stored about other types of biomolecules.

Generalizing Biological Molecules



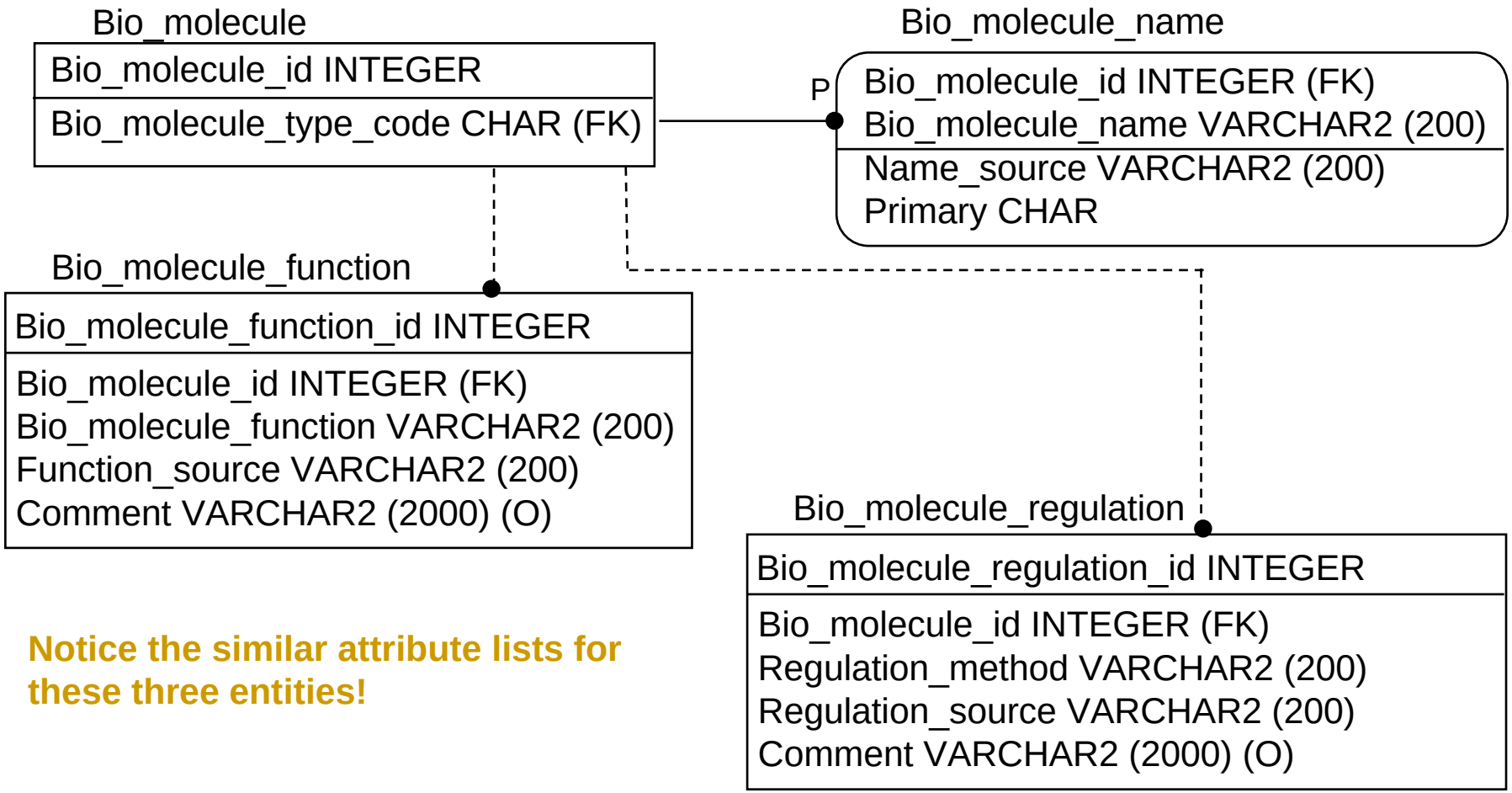
Without generalization, I would have needed one function table for each biomolecule type

Generalizing Biological Molecules



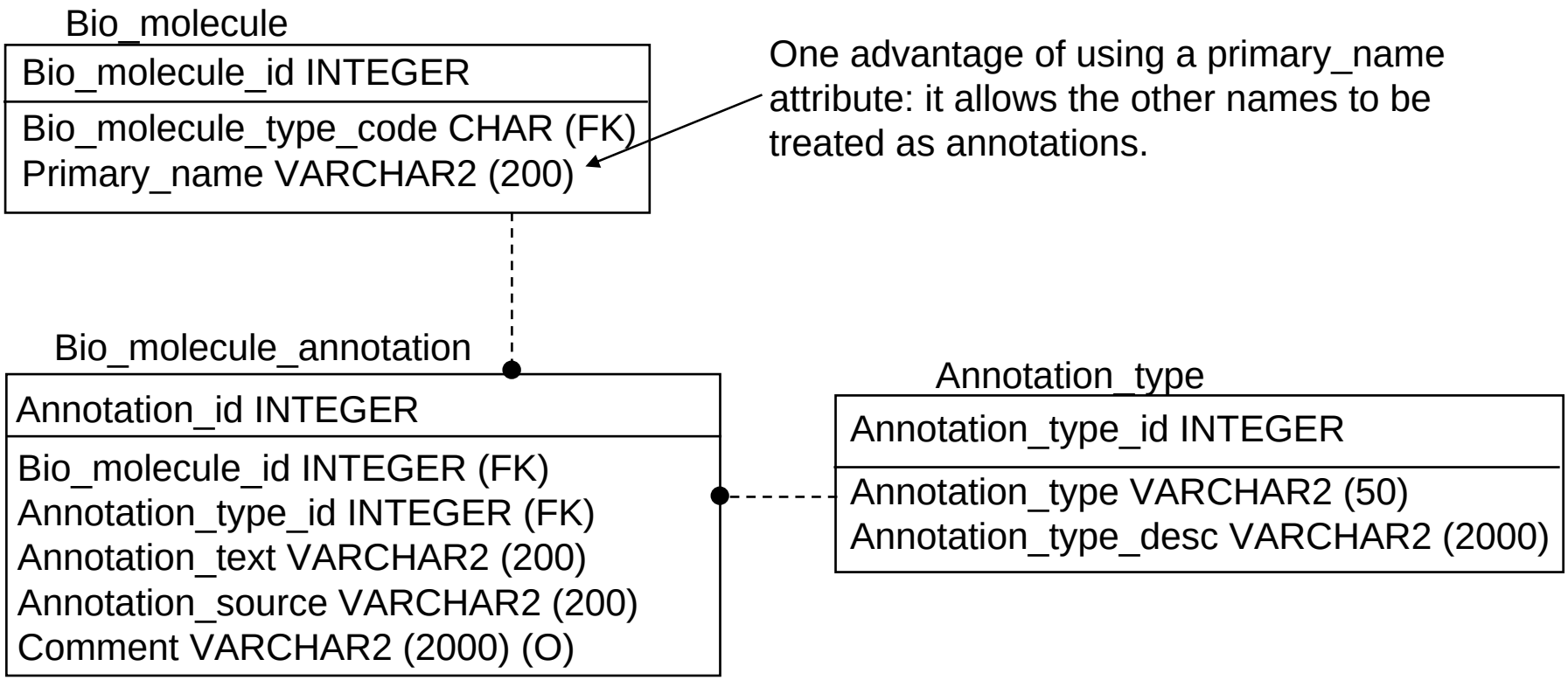
None of the relationships are required, because no type of info is stored for all biomolecule types

Further Generalization



Notice the similar attribute lists for these three entities!

Further Generalization



This design has two advantages:

- **Fewer entities**
- **Easy to add new annotation types**

Limits of Generalization

Bio_sequence

Bio_sequence_id INTEGER
Bio_molecule_id INTEGER(FK) Sequence_source VARCHAR2 (200) Source_db_sequence_ident VARCHAR2 (50) Sequence_text CLOB

Bio_molecule_structure

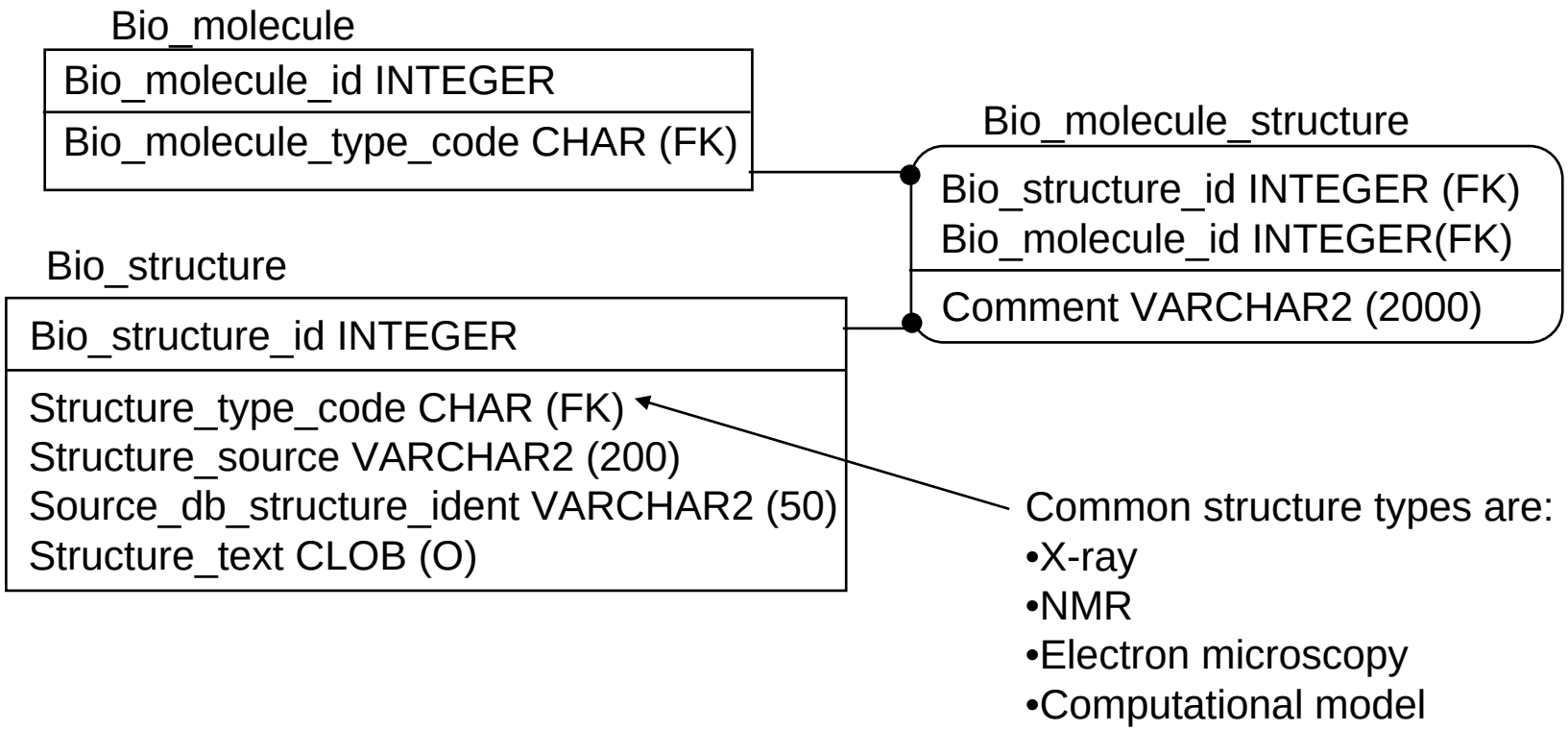
Bio_structure_id INTEGER
Bio_molecule_id INTEGER(FK) Structure_type_code CHAR (FK) Structure_source VARCHAR2 (200) Source_db_structure_ident VARCHAR2 (50) Structure_text CLOB (O)

Bio_sequence and Bio_molecule_structure have almost indentical attribute lists, should we generalize them into a supertype?

There is no “right” answer, but I wouldn’t. The supertype seems forced. Could use “Bio_molecule_structure”. Sequences are primary structure, three-dimensional structures are tertiary structure. But would we store secondary and quaternary structure data, too?

Also, there can actually be more than one biomolecule in a “structure”: this is not true of sequences.

More Detailed Storage of Biomolecule Structure



Respecting Scientific Culture

- Some important aspects of scientific culture:
  - Need to track the source of data
  - Need to accommodate the “fuzziness” of biology
- Databases that fail to respect these aspects will not be used

Tracking the Source of Data

- Discussed in slides for week 3
- Source is often provided by a peer-reviewed publication
  - Publication citation provides unequivocal link to the source
- Common to simply provide a link to PubMed
- Some sources are not in PubMed, and must be handled differently
- Merely storing the PubMed identifier does not allow queries into the reference data

Handling "Fuzziness" in Biology

- Recognize that there will be an exception to any biological classification scheme
  - Make schemes user-extensible using "lookup" tables
  - Provide a comment field so that users can document the exceptions
- Accommodate uncertainty in biological data

Accommodating Uncertainties

- Uncertainty is associated with all scientific data
  - Imperfections in measurement techniques
  - Incomplete knowledge
- Methods to handle uncertainty are chosen based on:
  - Type of uncertainty
  - Requirements of the scientists using the data
- Ignoring uncertainty can corrupt your database
  - Scientific conclusions may be based on data in DB
  - Uncertainty in data will influence conclusions
  - If users can't assess uncertainty of data, your database loses value

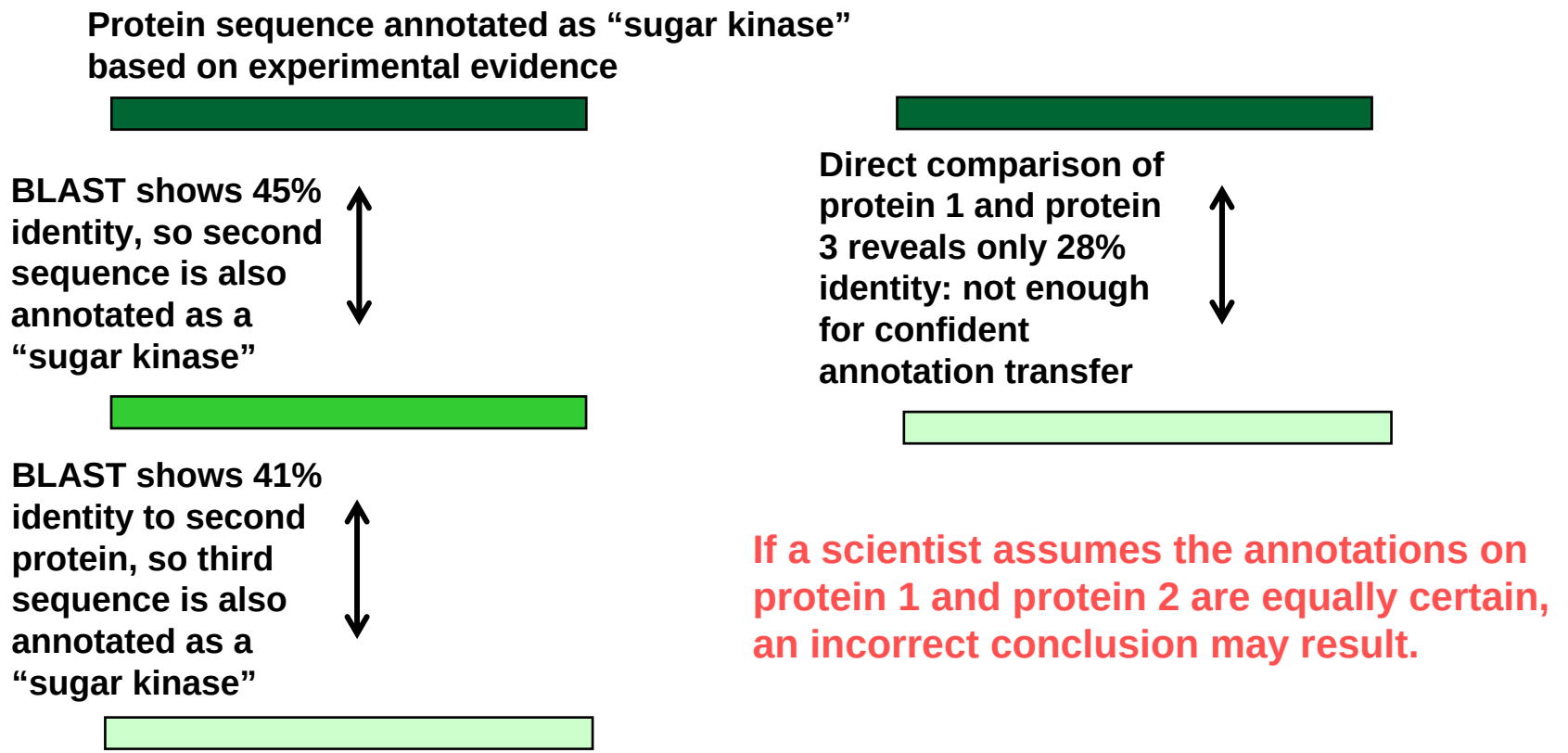
Types of Uncertainty

- Uncertainty in quantitative data can be calculated
  - Store raw data, and calculate on the fly
  - Store data and calculated error (e.g., average and standard deviation)
- Uncertainty in qualitative data is more difficult to handle
  - Some types of experiments are inherently less certain than others

Examples of Biological Data with Uncertainty

- Protein-protein interactions
  - Large scale studies and individual studies have differing uncertainties
- Biophysical measurements
  - Often include quantitative uncertainties
- Protein function annotation
  - Large difference in uncertainty between experimental and computational annotations

Function Annotations: Example of the Need to Include Uncertainty



If a scientist assumes the annotations on protein 1 and protein 2 are equally certain, an incorrect conclusion may result.

Including Uncertainty on Annotations

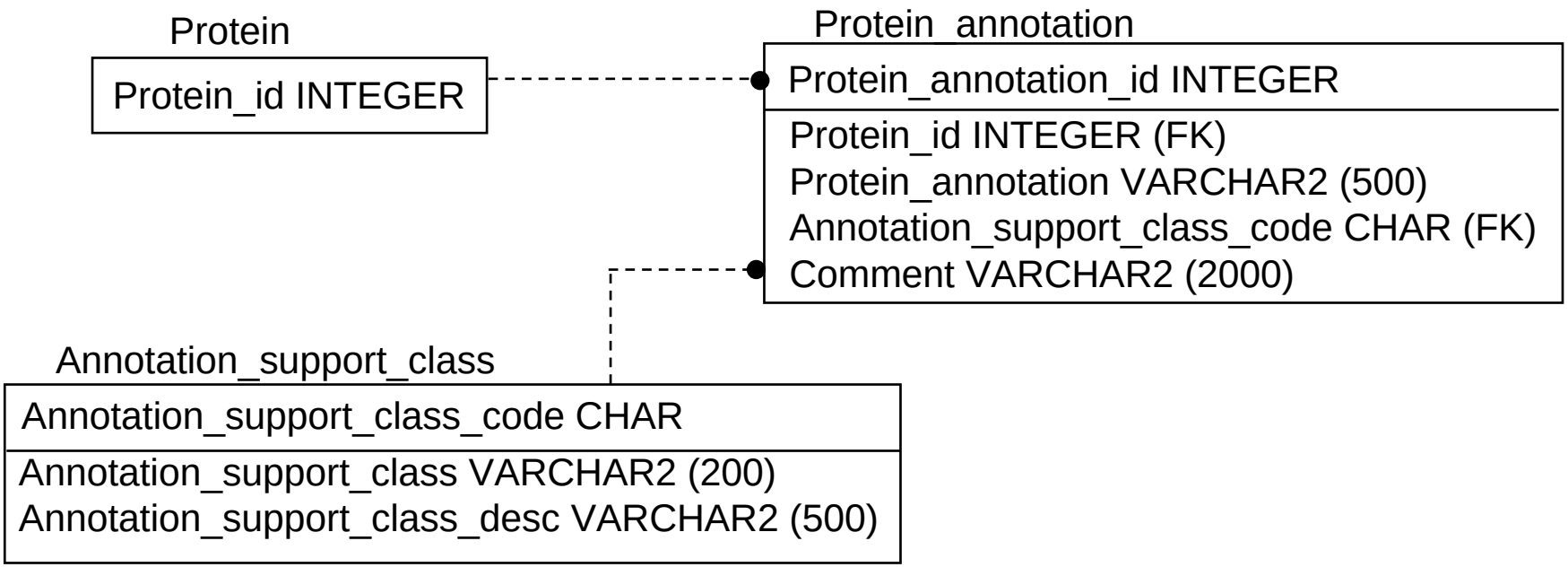
- The problem illustrated by the example is not caused by the first annotation transfer
- The problem is caused by the fact that a scientist using the data does not know that the annotation on protein 2 is less certain than the annotation on protein 1
- Solution is to include this uncertainty in the data presented to the user

Including Uncertainty on Annotations

- Include it in the annotation text: annotate protein 2 as “sugar kinase (by similarity)”
  - GenBank and other big sequence databases do this
- Include information about the source of the annotation: classify annotation on protein 1 as “experimental” and annotationon protein 2 as “computational” or “derived”
  - Gene Ontology includes evidence classifications
- Link annotation directly to the supporting data
  - May be appropriate for database for lab/company that is in the protein annotation business

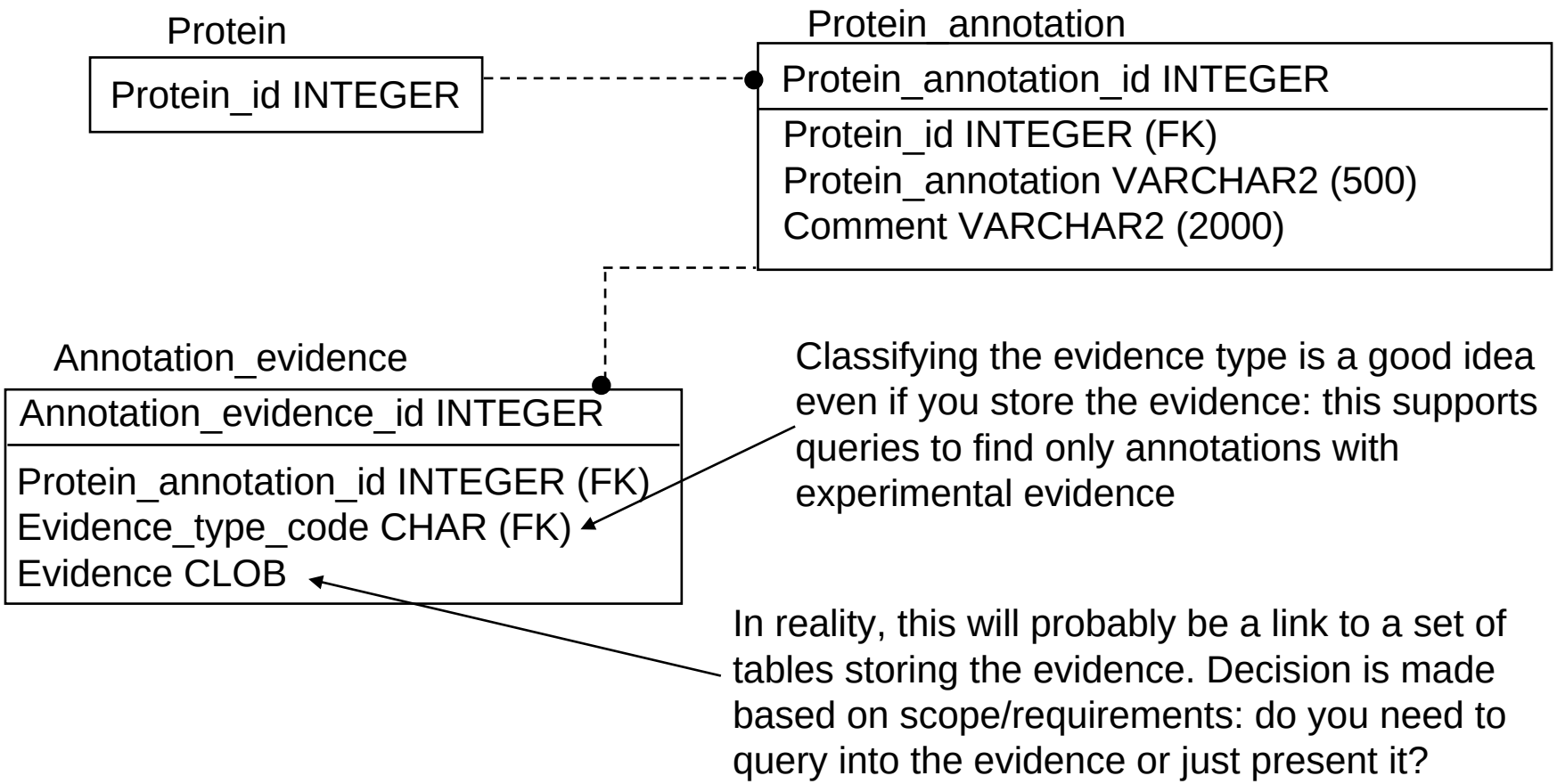
Including Uncertainty on Annotations

Using the Classification Method:



Including Uncertainty on Annotations

Storing the Supporting Evidence:



Handling Complex Data Types

- Two types of complex data types are common in biological databases
  - Data that can be broken into normalized tables, but is often used in a flat file format
  - Data that is truly complex, and cannot be broken into tables
- Both are often left outside of the database or stored in a CLOB or BLOB field
- However, there are different considerations in the handling of the two types

Flat File Formats in Biology

- Nucleotide and protein sequence data
  - GenBank
  - SWISS-PROT
  - FASTA
- Protein structure data
  - PDB
- Gene expression data
  - MAGE-ML (not really a flat file, an XML format)

Handling Flat File Formatted Data

- Decision to be made: parse into the database or store in file system and reference from the database?
- Answer depends on:
  - Scope of the database
    - If data in the flat files is not in scope, it can be difficult to justify effort required to parse it into the database
  - Resources available for handling data
    - Flat file-based systems have fewer integrity constraints, can change with little notice, and often have exceptions.
    - Can lead to frequent revisions of the parsing software
  - Politics of your organization
    - Some bioinformaticists may prefer flat files because that is what all of their tools run on

Deciding Whether or Not to Parse

- Advantages of parsing data into database tables
  - More thoroughly integrates data
  - Allows more complex queries on the data
- Advantages of leaving data in flat files
  - Don't have to handle inconsistencies and changes in flat files
  - Data is stored in the format required by many bioinformatics tools

Complex Data Types in Biology

- Complex data types are the raw data for a variety of experiments
  - Images (Gene expression arrays, microscopy)
  - Spectra (NMR, mass spec.)
  - Electron density (X-ray crystallography, electron microscopy)
- Complex data is usually further analyzed
- The results of these analyses are often stored in databases
- The raw data may be stored as a BLOB, or stored in the file system and referenced

Storing Data in Large Object Fields

- Both flat files and complex data types can be stored in large object fields (LOBs) or in file system (database references path)
- Disadvantages of storing data in LOBs
  - Often, data must also exist outside of DB for access by specialist programs
    - Storing data in two places can lead to inconsistencies
- Disadvantages of storing data in the file system:
  - Data is outside of DBMS consistency controls
    - If data is changed, references from DB may no longer be appropriate or correct
  - Data can be moved, making DB references stale

Parting Words

- Successful design of biological databases requires understanding of biology and database principles
  - Will not necessarily have both in same person
  - Work in teams, and respect the complexity of the field that is not your own
- For most research-scale DBs, performance will be adequate without any tricks
  - Don't fall into the trap of denormalizing because you've heard that normalized databases have performance issues
  - Denormalize only as a last resort
- For many of the design issues, there is no "right answer"
  - Decisions often depend on requirements of the DB
  - Field is too young for consensus on best way to handle data
  - Don't get "analysis paralysis": take your best shot, and learn from how it works (or doesn't!)