­­­­­­­Databases – DB Models

2.1 Data Models

* [SLIDE 1] What is a data model? It is a pattern or notation for describing data.
  + A description of how to conceptually structure the data, what operations are possible on the data, and any constraints on the data.
  + Analogous to creating a class in C++.
  + **Structure** – how we view the data abstractly. What information do we need to record about the items we are storing in the database? when creating a class, we think about the fields/variable that go into a class.
  + **Operations** – what is possible to do with the data? In DB parlance, these are often called queries. class=methods.
  + **Constraints** – how can we control what data is legal and what is not? Class=data types, more sophisticated for DB
* [SLIDE 2] Most important data model for DBs is the RELATIONAL model.
  + Structure: relation OR table
  + Operations: relational algebra OR SQL
    - Select certain rows or certain columns that have or do not have certain properties.
    - This limitation is actually a good thing. By using a language like relational algebra or SQL, this allows the DB to optimize our queries to make them as efficient as possible.
    - For instance, when you want something in a database to come out sorted, you literally add the phrase ORDER BY into your query. This allows the DB to pick them best way to do the sort. On the other hand, in python or C++, if you want to sort something, in general, you have to pick the algorithm, and no compiler is smart enough to replace your bubble sort with a quick sort.
  + Constraints: can enforce restrictions like grade must be in the set ABCDF
* Show harry potter table.
  + Structure is tabular, you can think of this as an array or list of structs. Each row represents the values of one entry in the array of structs. Behind the scenes, this probably isn’t an array or list, but some more sophisticated data structure. (B-tree)
* [SLIDE 3-5] Semistructured model is an alternative – usually resembles graphs or trees. = data that Is still “structured” but not in the relational format.
  + Structure is XML or JSON.
  + Operations usually involve following paths in the hierarchy (XQuery)
  + Constraints: document type definitions (DTDs)
  + Other data models – object databases, or object-relational
    - Graph databases
    - NoSQL, NewSQL [document databases, key-value store,
  + Object-relational – similar to relational, but values in a table can have their own structure, rather than being simple strings or ints or floats (they can be objects)
    - Relations can have methods associated with them
* NoSQL
  + types.
    - document (stores a whole structured document)
    - key-value store (basically a huge lookup table) (think hashtable, but doesn’t necessarily use hashing).
    - wide-column store (2d key-value)
  + Sacrifice consistency, but offer “eventual consistency” – in that changes are propagated often within milliseconds.
* SLIDE 6: Relational model is most common
  + Simple: built around a single concept for modeling data: the relation or table.
    - A relational database is a collection of relations.
    - Each relation is a table with rows and columns.
    - An RDBMS can manage many databases at once.
  + Supports high-level programming language (SQL)
    - Limited but useful set of operations.
  + Has elegant mathematical theory behind it.
* SLIDE7: Terminology [start 2.2] **WRITE TERMS ON BOARD WITH DEFS**
  + **Relation** == 2D table
    - **Attribute** == column name
    - **Tuple** == row (not the header row)
  + **Database** == collection of relations
* [SL8] Relation terminology
  + A relation includes two parts:
  + The **relation** **schema** defines the column headings of the table (attributes/fields)
  + The **relation** **instance** defines the data rows (tuples, rows, or records) of the table.
  + Schema is kind of like defining a struct in C++. You define what members the class has and what their data types are.
  + Each row in a DB is an instance of the class, so an instance of the entire relation is a collection of these rows.
* [SL9] Schema
  + A schema is written by the name of the relation followed by a parenthesized list of attributes.
  + Grades(First, Last, Course, Grade)
  + A **relational database schema** is the set of schemas for all the relations in a DB.
* Tuples.
* Domains
  + A relational DB requires that every component of a row (tuple) have a specific elementary data type, or **domain**.
  + string, int, float, date, time (no complicated objects!)
  + Grades(First:string, Last:string, Course:string, Grade:char)
* Relation equivalence
  + Relation is a ***set*** of tuples, not a list. [that is, there is no order to the rows, though in the real world, there often is]
  + Attributes in a schema are a ***set*** as well. [ no order to cols either]
  + However, the schema specifies a "standard" order for the attributes.
  + How many equivalent representations are there for a relation with *m* attributes and *n* tuples?
    - How many different ways could I draw this table and still technically have the same relation?
    - Answer = m! \* n!
* Degree/cardinality
  + **Degree/arity** of a relation is the number of attributes in a relation.
  + **Cardinality** is the number of tuples in a relation.
* Keys
  + Keys are a kind of **integrity constraint**.
  + A key is a set of attributes that have the following property.
  + A set of attributes K forms a key for a relation R if:
    - no pair of tuples in an instance of R may have the same values for all attributes of K.
  + What are keys of Grades? (can’t tell just by looking at the table)
  + However, **probably**, keys might be (first, last, course).
  + Why is (first last) almost certainly not a key? students enrolled in multiple classes.
* Artificial keys [slide with 3 tables]
  + Often, many real world databases create artificial keys, because often we want to be 100% sure that we will never have any duplicates in combinations of attributes.
  + artificial key = completely made up piece of information that serves no other purpose than to identify a tuple in a relation.
  + What are the keys in these relations?
    - ans: Key for student table = SID. Key(courses) = CRN or maybe <CRN,Sem,Year> Key(grades)=all attribs.
  + What are some examples of artificial keys in the real world?
    - R-number, SSN, Drivers license number, produce codes (really all bar codes), every book published has an ISBN
    - sometimes, people use artificial key to mean only a key that has no meaning outside of the DB. So by that stricter definition, all of the things we just listed aren’t really artificial keys, because they are used in the real world by people.
* Let's expand these relations to handle the kinds of things you'd like to see in WorkDay.
* Keep track of students, professors, courses, who teaches what, enrollments, pre-requisites, grades, departments & their chairs.
  + Only one chair per department.
  + Student cannot enroll in multiple copies of the same course in one semester.
  + Other constraints that are logical.  
      
      
    write schema for past 3 tables on board
  + develop new schemas
  + Profs(??) develop schema:
  + If CRN is always unique, where do we store who teaches a course? (in courses tbl)
  + If CRN is not unique, where do we store who teaches a course? (in sep new tbl)
  + Chairs(??)