**Database Design & Administration - CST8250**

**Week 4 - February 1st**

Class Intro

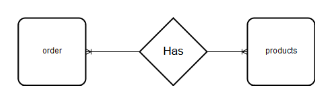
* You have a test coming up on February 22nd
* The midterm will cover lectures 1-5, no webcam necessary
* February 15th will be a class for an overview/review for the test
* “Please don’t cheat <3” - Dan
* Last set of Labs due before the break are due on the February 18th

Overview

* Resolving relationships
* Database design process
* Data types
* Keys: Natural vs Synthetic

Resolving Relationships

* One to One
  + No resolving necessary
* One to Many
  + Optional to resolve to one to one in rare cases, not common and likely undesirable due to information loss
* Many to Many
  + Resolving is almost always desirable
* Many to Many
  + Essentially we are dealing with two lists with duplicate data and cannot easily ask questions such as:
    - How many orders have been made?
    - How many products do we sell?



Resolving Relationships Example

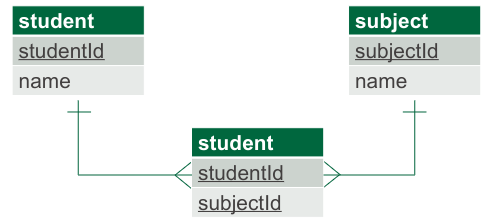
Many to Many

* Add an associative entity – a table that contains the keys of both tables



Many to Many

* Add another entity – a table that contains the keys of both tables

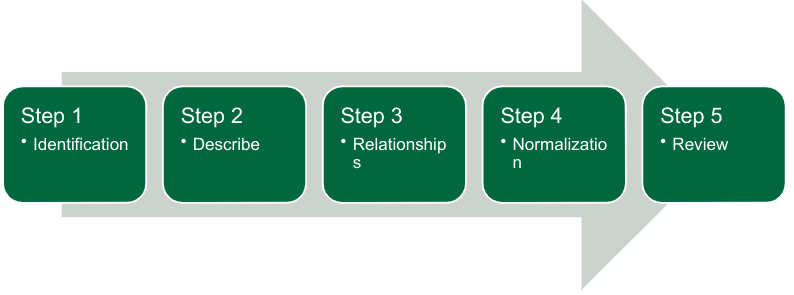


* The foreign keys from both tables are the primary key of the new table
* The bottom table is an associative entity

DB Design Process

* The design process is iterative
* There is no “perfect” design. Reaching for perfection usually causes more problems. You are reaching for “good enough” and “not stuck in a corner”
* Made up of 4 steps + an end of process review.

Design process steps



Step 1 - Identification

* Depending on the source of data two common paths are available.
  + Path 1 – Recreation/Reverse engineering
    - You are given piles of information (invoices, orders etc.) and you need to recreate a better version of that
  + Path 2 – Clean room implementation
* Common steps however include:
* Identify all possible gross data objects. i.e. users, customers, orders, etc.
* List the objects and categorize them

Step 2 - Describe

* With each of the objects add all the basic fields. Primary key, descriptor, etc.
* Try to identify as many fields/properties as possible.
* Assign data types to each of the fields.

Step 3 - Relationships

* Create connections between objects
* Identify which objects are parents and which are children
* Identify which relations are mandatory
* Create the foreign keys as needed (remember the naming conventions)

Step 4 - Normalization

* Using the rules of normalization break down the design. The rules are:
  + 1st NF – tables should have no repeating fields, a primary
  + key and organized in rows
  + 2nd NF – All data in tables that have correlated PKs must depend on the whole key
  + 3rd NF – All data must rely on the PK.
* Create reference tables as needed
  + Also known as a lookup table, like the source of a drop down menu on websites (Titles, Countries, Shipping methods etc.)
* Replace fields in standard tables with reference table FKs.
* Goal is to normalize to 3rd NF.

Step 5 - Review

* Review the design for potential issues
* Try to find a peer to review your design
* After identifying weaknesses, start over at step 1 to implement the changes into the current design.
* Also review with an eye to the future and try not to over engineer.
  + Programmers tend to take short cuts or do way more than is necessary

Why is test data important?

* Testing is important.
* Gives developers something to work with.
* Load testing
* Many sites now offer this service.
  + generatedata.com – can download and install into own webserver
  + mockaroo.com – not free, different options from generatedata
  + freedatagenerator.com – Supports parent-child structures

Data Types

* Text types
  + **Char**/character – fixed length string, always occupies defined space
  + **Varchar**/character varying – variable length string, occupies string length + 1
  + **Text** – used to large chunks of text – MySQL has 3 different types.
* Number types
  + Integers – whole numbers. INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT
  + Decimal/numeric
  + Float/double
  + Bit
* Date and Datetime (contains a date and time)
* Timestamp – special version of datetime that respects timezones
* Time and Year

Choosing Data Types

* When choosing your data types, take into consideration:
* How big is the data?
* Is it numeric? Decimal places?
* If it’s a date, should you include the time
  + The answer is always yes except in certain cases like date of birth unless twins
* How big is the text
* Just say no to BLOBS.
  + Because they are database cancer
  + Unless you want to store a different text type like a different alphabet

Synthetic vs. Natural Keys

* Composite key. A key that is composed of two or more attributes.
* Natural key:
  + A key that is formed of attributes that already exist in the real world.  For example, U.S. citizens are issued a Social Security Number (SSN)  that is unique to them (this isn't guaranteed to be true, but it's pretty darn close in practice).  SSN could be used as a natural key, assuming privacy laws allow it, for a Person entity (assuming the scope of your organization is limited to the U.S.).
* Synthetic key (A.K.A. Surrogate key):
  + A key with no business meaning.
* Primary key:
  + The preferred key for an entity type.
* Foreign key:
  + One or more attributes in an entity type that represents a key, either primary or secondary, in another entity type.

Issues with Natural Keys

* Con 1: Primary key size – Surrogate keys generally don't have problems with index size since they're usually a single column of type int. That's about as small as it gets.
* Con 2: Foreign key size - They don't have foreign key or foreign index size problems either for the same reason as Con 1.
* Con 3: Esthetics - Well, it’s an eye of the beholder type thing, but they certainly don’t involve writing as much code as with compound natural keys.
* Con 4 & 5: Optionality & Applicability – Surrogate keys have no problems with people or things not wanting to or not being able to provide the data.
* Con 6: Uniqueness - They are 100% guaranteed to be unique. That’s a relief.
* Con 7: Privacy - They have no privacy concerns should an unscrupulous person obtain them.
* Con 8: Accidental Denormalization – You can’t accidentally denormalize non-business data.
* Con 9: Cascading Updates - Surrogate keys don't change, so no worries about how to cascade them on update.
* Con 10: Varchar join speed - They're generally int's, so they're generally as fast to join over as you can get.

Disadvantages of Synthetic Keys

* Getting The Next Value – Most servers support auto incrementing of some sort.
  + This one is not really a valid argument
* Extra Indexes - if you have N indexes per table in natural key world, you’ll always have N + 1 indexes in the synthetic/surrogate key world