Ch. 3 - class notes

attribute = column

I will design by:
The Key
The whole Key
Nothing But the Key

1NF: no multi-valued columns
2NF: every attribute depends on logical key
3NF:

* Design db by BCNF *

- you'll hardly ever use or see 4NF / 5NF

Database Ch. 4 - notes

I. Assess Table Structure

- When told to create a db, first look at the data table's structure and its content
  - determine functional & multivalued dependencies, candidate keys, and each table's PK
  - assess validity of assumed referential integrity constraints
    - to test, find a column that may be a foreign key & select all that aren't in the inner join of the 2 tables w/ the columns
      - if no results, then no violation of ric

good test for inconsistent values

II. Designing Updatable Db's

- these db's are usually the operational db's, like an online transaction processing system

Advs & Disadvs of Normalization

Adv

- no mod anomalies
- reduce dupl data
- no data integrity problems
- save file space
- single-table queries run faster

Disadv
- harder SQL required for multitable subqueries and joins
- could mean slower apps if more work for DBMS bc they have to read more tables

Functional Dependencies
- all tables need to be in BCNF

Normalizing w/ SQL

- if a table isn't in BCNF, make 2 tables & try again
- BCNF: all determinants (col A is dep. on col B) are candidate keys (all cols in table1 dep. on col A)
- SQL stmts are easier

Ex. of SQL for referential integrity constraint
- Table1.col2 must exist in Table2.col1;

Choosing Not to Use BCNF

- possible scenarios: zip codes
- if a table is not in BCNF, it is susceptible to mod. anomalies and inconsistent data
- So, if the data in your table is never updated & inconsistencies are easy to correct, you don't have to do BCNF form

Multivalued Dependencies = SERIOUS

- should always be eliminated (from updatable tables)
- the SQL complexity needed to deal w/ these anomalies is insane

III. Designing Read-Only Db's

- used by BI Systems for assessment, analysis, planning, and control
- used in data warehousing
- normalization not usually used in read-only db

Denormalization

- data for read-only is commonly extracted from operational db's
- the data you're given will most-likely be normalized
  - join together the separated tables by running a query join & saving the result as a table

Customized Duplicated Tables

- copies of the data are made & tailored for special instances

IV. Common Design Problems (when designing from existing data)

4 problems: 1.) the multivalue/multicolumn problem
2.) Inconsistent values
3.) Missing values
4.) General-purpose remarks column

Multivalue/Multicolumn Problem

- storing multiple columns of the same thing (Address1, Address2, etc.)
  - problems occur when there's not enough columns (i.e. 10 addresses), or querying for a value that's actually null in a column
  - solution: make a new table instead of a limited number of columns
  - makes query easier & removes NULL values

- when deciding, think about how policies could change if you decided to keep 1 table. If ~~as~~ allowed cars went from 2 to 4, it'd require a db redesign. Think ahead.

Inconsistent Values

- occurs b/c there's different forms of the same data value (i.e. Large, largee)
- dont code the same entries differently!
- common when combining data
  - Solutions
    - query using Group By
    - test an inner join of the foreign key
      - bad values would stand out

Missing Values

- Null values can indicate:
  - value is inappropriate
  - value is appropriate but unknown
  - value is appropriate & known, but not here yet
- test by querying using IS NULL
  - check especially w/ foreign keys

NOTE: Null itself is an inconsistent value. Also check for: 'unknown', empty string, -1, blank string, et

General-Purpose Remarks Column

- ~~scar~~ scary bc it usually has needed data in these notes/comments that are nearly impossible to get out
- solution: id all purposes of the remarks column & create new columns for each purpose OR dont have the column at all (if your starting from ~~a~~ scratch)