# Databases (CSCI2141)

**Covers Lectures 2 to :**

Lecture Dates: January 10th, 12th, 17th, 19th, 24th, 26th, 31st, 2012
February 2nd, 7th, 2012

*Introduction, Review*

Chapters 1, 2 – Assignment 1 Posted **Thursday** (Written, straight-forward) – Do all questions (some questions may be out-of-order).

- Textbook is available in the bookstore.
  - A lot of online resources.
  - Read the 1st two chapters.
- Database
  - Computer structure. Shared, integrated that houses a collection of *related data*.
  - Evolved from manual, computerized file systems (e.g. in an office, a library).
- Different Types of Databases (Table 1.1)
  - We'll be using MySQL. On Bluenose.
  - There are other ones, i.e. MS Access.
- Uses
  - Inventory control.
  - Payroll.
  - Banking, financial systems.
  - Reservation systems.
  - Software development.
  - Telecommunication systems.
  - And so on.
- Recall: DBMSs – interface between user and inner workings of the database itself. Program (or set of programs) managing structure; controls access.
  - Developed to address file system weaknesses.
  - Builds a lot of security into the database.
  - In a file system, data stored in independent files (each requiring extensive programs, different departments).
    - Requires extensive programming.
    - System admin is complex, difficult.
    - Changing existing structures = difficult.
    - Security features inadequate.
    - Independent files tend to contain redundant data.
- See Figure 1.6
- What are data?
  - Data is plural. Raw facts – not useful for answering questions.
  - Requires context, attribute. Versus information. Example: 112468 (could be a date, salary, address).
  - ***Metadata***: *Data about data*. Characters, relationships.
  - ***Field***: A character, group of characters (numeric, alphabetic) that has special meaning. Used

to define, store data (example: C_PHONE, C_NAME, etc.).
    - ***Record***: Logically connected set of fields.
    - ***File***: Records together.
    - ***Query***: Question, task asked by end user of a database in form SQL code (specific request issues to DBMS). *Query Language*: Non-procedural; used by a DBMS to manipulate data. Non-procedural: lets users specify what must be done without having to specify how it is to be done. Request, what get back does not change if the inner workings change (black box).
    - ***Ad-hoc Query***: Spur-of-the-moment question. Advantage of DBMSs. Can just put it in.
    - ***Query result set***: Collection of data rows returned by a query.
    - ***SQL***: Composed of commands – enables users to create databases, table structure, etc.
    - ***Data inconsistency***: When different versions of the same data *appear in different places* in the same database. Example: J. Mason vs. Jane Mason.
    - **Data formats**: Logical vs. Physical. Logical: way in which humans view data. Physical: way computer stores it.
    - ***Data Redundancy***: Same data stored in different places unnecessarily. Real problem: tends not to all be changed when one is. Cause of data inconsistency.
    - ***Data Anomaly***: Not all required changes in redundant data are made successfully.
    - ***Structural Dependence***: Access to file is dependent on its own structure; all file system programs must be modified to conform to a new structure (for example: a new field is added).
    - ***Structural Independence***: File structure can be changed without affecting application program's ability to access data.
    - ***Data Dependence***: Access changes when data storage characteristics change (integer to decimal for instance).
    - ***Data Independence***: Not affected by changes.

**Chapter 2: Data Modeling**

- First step in DB design.
    - Different views of same data; data modeling reduces complexities of DB design. Blueprint for DB design.
- Data Models – relatively simple, usually graphical, representations of complex real-world data structures.
    - Data modeling is an iterative, progressive process.
    - Will almost always be more than one "correct" solution. *Will meet the needs of the end-user*. Can be tricky. All users have different views.
- Components of a Data Model
    - Description of data structure.
    - Set of enforceable rules.
    - Manipulation methodology.
- Importance
    - Facilitate interaction among designer, applications programmer, end-user.
    - End users have different views, needs for data.
    - Data model is an abstraction. Cannot draw required data out.
- Basic Building Blocks

- Entities, Attributes, Relationships, Constraints
- ***Entities***: Anything about which data are to be collected, stored (person, place, thing, concept, event, etc.). Each occurrence is unique, distinct (customer, product, etc.).
- ***Attributes***: Characteristic of an entity. Has name, data type. Equivalent of fields in file systems.
- ***Relationships***: Relationship among entities, association. Three types: 1:M (one to many), M:N/M:M, 1:1 (less common).
  - One-to-many (1:M) – associated with many instances of the related entity.
  - Many-to-many (M:N) – one associated with many occurrences of a related entity, and one occurrence of the related entity is associated with many occurrences of the first entity.
- ***Constraints***: Restriction placed on data. Normally expressed as rules. Help ensure data integrity.

- Business Rules
  - Brief, precise, unambiguous description of a policy, procedure, principle within a specific organization. Basically a rule.
  - For example pilot cannot be on duty for more than 10hr in a 24hr period.
  - Help create, enforce actions within organization's environment. In writing (ideally), updated to reflect change. Used to define entities, attributes, relationships, constraints.
  - Should be easy-to-understand; main and distinguishing characters as viewed by company. Help standardize company's view of data. Allows designer to understand nature, role, scope of data.
    - All the designer to understand business processes.
    - All the designer to develop appropriate relationship participation rules and constraints and to create an accurate data model.
  - Will not always give everything you need. Sometimes could be conflicting.
  - Sources include company managers, policy makers, department managers, written documentation (procedures, standards, op manuals), direct interviews with end users.
- Translating Business Rules into Data Model Components
  - Noun in a business rule will translate into an entity in the model.
  - Verb associating nouns will become a relationship. Relationships are *bidirectional*. Two questions to identify the type:
    - How many instances of B are related to one instance of A?
    - Vice versa.
  - Note that not all business rules can be modeled.
- Naming Conventions
  - Naming occurs during translation of business rules to data model components.
  - Should make objects unique, distinguishable (think variables). Be descriptive. Facilitates communication between parties and promotes self-documentation (semantic).
  - For example: capital letters are typically used in entities.
- Review
  - Data modeling is an iterative, progressive process.
  - Data modeling is the first step in database design; an abstraction of a complex real-world data environment.
  - Basic Data Modeling Components: Entities, Attributes, Relationships, Constraints.

- Use business rules to identify, desfine basic modeling components.
- Used to define entities, attributes, relationships, constraints.

*The Evolution of Data Models* (Chapter 2, cont'd)

- ***Hierarchical, network data models*** were early data models that are no longer used, but some concept sfound in current data models.
  - 1960s – manage large amounts of data for complex management projects such as the Apollo rocket.
  - Had to create new systems to handle this.
  - Idea of a hierarchy, a tree. Parents, children – each parent can have many children, but each child can have only one parent (1:M relationship).
  - So: a lot of replicated information.
  - ***Structure contains levels, segments***. Segment: equivalent to a record type.
- Network model.
  - Improved database performance, imposed standard. Allowed records to have more than one parent.
  - Collection of records in 1:M relationships.
  - Generally not used today. Some concepts still used.
  - Concept still used today: Schema.
    - Overview of entire database.
    - Logical grouping of database objects, related to each other.
    - Subschema: portion of database "seen" by programs.
  - Concept still used today: DML (Data Manipulation Language) – example: SQL. Gives set of commands that allow you to manipulate data. DDL (Data Definition Language) – example: SQL does this, too. Allows to define schema, etc.
  - Became very cumbersome as information grew. Did not have ad hoc query method. Had to generate code for even simple reports.
  - Structural change in databases could produce havoc in all application programs.
- ***Relational model*** is the current database implementation standard. What this course will focus on; what is used most commonly.
  - 1970 – was considered ingenious at time, but impractical. Not enough computer power.
  - Fortunately, power grew exponentially.
  - A relation (table) can be thought of as a matrix composed of intersecting rows, columns. Each row is called a tuple (like a record). Each column is an attribute (like a field).
  - Implemented through a RDBMS. Performs basic functions in addition to many more functions that make model easier to understand, implement. Most important advantage: hides complexity of model. Manages all physical details (not stored in actual tables).
  - Relational tables store colections of related entities. Related to one another through sharing of a common attribute (value in a column). One column can provide a link.
  - Relational diagram – representation of entities, etc.
  - For most RDBMS, query language is SQL. Data can be retrieved with less effort than with

any other database or file environment.
    - SQL-based relational database application involves:
        - User interface.
        - Set of tables.
        - SQL "engine" which executes queries.
- Entity relationship model is a popular graphical tool for data modeling that complements the relational model. Object-oriented model uses objects as the basic modeling structure.
    - Graphical tool for doing data modeling.
    - Introduced in 1976. Quickly became accepted standard.
    - ER models represented in a diagram – graphical representation of components.
    - Each entity instance (or occurence) is row in table. Entitiy set is collection of like elements.
    - Connectivity: classification of relationship (1:M, etc.).
    - Chen Notation, Crow's Foot Notation (1: ||, Many: Crow Foot).
- ***Object-Oriented Data Model***: Basis for OODBMS. Semantic data model (semantic: meaning). Uses objects as the basic modeling structure.
    - Object: Contains both data, relationships in a single structure.
    - Resembles an entity in that it includes facts that define it.
    - See slides. Typically depicted using Unified Modeling Language (UML) class diagrams.
- Relational model has adopted many object-oriented extensions to become the extended relational data model (ERDM).
- XML databases manage unstructured data within a native XML format.
- See Table 2.1: Evolution of Data Models.
    - File system (1960s – 1970s) to Hierarchical and network (1970s) to Relational (Mid-1970s to present) and where Object-oriented, Object/relational (O/R) are more common (Mid-1980s to present).
    - Prediction in future: more hybrid databases. For example: O/R and XML.
        - Retain advantages of relational model. Provide OO view of underlying data.
        - SQL data services: cloud computing.
- Extended RDM: A hybrid model.
    - Semantic data model.
    - Includes many of OO model's best features.
    - Often described as an O/R DBMS.
- XML emerged as the de facto standard on the Internet.
- *See Figure 2.5* – evolution diagram.
- Common Characteristics
    - Conceptual simplicity & semantic completeness.
    - Each new model capitalized on shortcomings of previous models.
    - Some models better suited for some tasks.
- Degrees of Data Abstraction
    - DB Designer starts with an abstract view of overall data environment. Adds detail as come closer to implementation.
    - Using levels of abstraction can be helpful in integrating multiple views of data (sometimes conflicting).
    - 1970s – ANSI-SPARC defined as framework for data modeling based on three degrees: external, conceptual, internal. Fourth degree (very low): physical.

    - Figure 2.6 – Data Abstraction Levels. High level (end-user) to low-level (physical).
- External Model (High-Level)
    - End-user view of data environment.
    - Specific representation: external schema.
    - See Figure 2.7.
- Conceptual Model (A little lower)
    - All external views into a single global view. Combined in a single diagram.
    - Conceptual schema.
    - Independent of both software, hardware. Does not depend on DBMS software.
- Internal Model ("Seen" by the DBMS)
    - Maps conceptual model to DBMS.
    - Internal schema.
    - Software dependent, logically independent (can change without affecting conceptual model), Hardware independent.
- Physical Model (Way data saved on storage media).
    - Hardware, software dependent.
    - Relational model is aimed largely at logical level (physical-level details not required).
    - Physical independent when changes do not affect internal model.
- See Table 2.4: shows independence, etc.

*The Relational Database Model* (Chapter 3)

- Review – Hierarchical Model
- Review – Network Model
    - More than one parent.
    - *Perceived by user* as 1:M relationships, but way implemented (data structure) is using sets, has multiple parents.
    - Will not be tested too harshly on this.
- Review – Relational Model
    - Current standard.
    - OO has not taken over because expensive to take it over. One reason why extended, OO are not as common as they could be.
    - End user perceived data as being stored in tables (row in a relation – tuple, column – attribute).
- See Figure 2.1
    - Not considered data redundancy – not unnecessary; used as a link (***controlled redundancy*** – necessary to work).
    - For example: Agent Code.
- Review – Entity Relationship Model
    - Chen, Crow's foot notation.
    - In Crow's Foot Notation – one side is indicated by two short lines, many side as a three-pronged segment (bird's foot).
- Review – Object-Oriented Model
- Review – Newer data models.
    - Expensive to do upgrades.

    - Expensive to do changes to databases.
    - Slower process than what people may expect.
- Review – Data Abstraction
    - Data-modeling requirements are a functino of different data views, abstraction levels.
    - Three levels (External, Conceptual, Internal); fourth level – physical.
- Structural, data independence – means we can focus on logical structure (how humans view data) without considering physical aspects.
    - Data independence: data access unaffected by changes in physical storage characteristics (e.g. Integer, decimal).
    - Structural independence: changes in schema do not affect data access (e.g. Adding, removing attributes or entities).
    - See table.
- Logical simplicity: can think of related records as being in independent tables; simpler, more effective design.
- Tables
    - Logical view of relational database is based on logical construct known as a relation (from set theory).
    - Persistent representation of a logical relation (permanently).
    - Only one component in a database (an entity set). Database also holds metadata (relationships between entity sets).
    - Contains a collection of related entity occurrences (entity set). Often used interchangably with relation and table.
    - Each table row (tuple) represents a single entity occurrence. Each column = attribute (distinct name). Each intersection = data value.
    - Order of rows, columns is immaterial. All values in a column = same data format. Each column has a specific range of values (attribute domain).
- Data Types
    - Numeric
    - Character (text, string)
    - Logical (true/false, yes/no)
    - Date (stored in Julian date format - represents a problem; 5-digit number (2-digit year, 3-digit day-of-year – if year digits <30, assumed to be a 2000 century year; if >= 30, assumed to be a 1900 century year).
- Each table must have an attribute or combination of them (primary key) that uniquely identifies each row. Also used to establish relationships among tables, ensure integrity of data.
    - See Figure 3.1
    - STU_NUM is the unique identifier for first column.
- Keys
    - Consists of one+ attributes that determine other attributes.
    - Example: invoice number identifies all of the other invoice attributes.
- Determination
    - If A determines B, if you know value of attribute of A, you can look up value of B.
    - A -> B or A -> B,C,D
- Functional Dependence
    - Attribute B is functionally dependent on A if each value in A determines one and only one

value in column B.
    - Ex: STU_PHONE is functionally dependent on STU_NUM. STU_NUM not functionally dependent on STU_PHONE.
- Composite Keys
    - Made up of more than one attribute (to define functional dependence).
    - Any attribute part of it: key attribute.
    - Example: last name, first name, initial, phone number probably a good composite key in the STUDENT table.
- Full Functional Dependence
    - If attribute B is functionally dependent on a composite key A but not on any subset, attribute B is fully functionally dependent on A.
    - Essentially, *minimum key* needed.
    - Superkey
        - Any key that uniquely identifies each row.
        - Determines all of a row's attributes.
        - Does not have to be minimally; as many attributes as you want as long as it uniquely identifes.
    - Candidate Key
        - Minimal superkey.
        - Full functional dependence.
        - STU_NUM by itself is a candidate key.
        - The ***primary key*** is a candidate key; therefore is also a superkey.
        - Entity intengrity: Each row is uniquely identified by the primary key.
- ***Entity Integrity***
    - Means a proper search for an existing row will be successful, otherwise, does not exist.
    - A null is not permitted as the primary key.
        - Null: No value at all.
        - Does not mean a zero, a space; can never be part of a primary key.
        - Should be avoided in other attributes. Sometimes difficult to avoid.
- ***Null Values***
    - Are confusing. Is it unknown, missing, condition N/A?
    - Can also create problems when funcitons such as COUNT,AVERAGE,SUM are used.
    - Can also create logical problems.
    - To avoid them, can use special codes known as ***flags*** (indicate absence of some value; e.g. -99).
- Controlled Redundancy
    - *Necessary* redundancy; in order for relational databases to work.
    - See above.
    - See Figure 3.2.
    - Can have relational diagram (3.3).
- Foreign Keys
    - A ***foreign key*** (FK) is an attribute (or combination of them) in one table *whose values match the primary key* values in a related table (or whose values must be null).
    - Referential integrity: if foreign key contains a value, that value refers to a row in another relation. Makes impossible to assign a non-existing foreign key.

- ***Secondary Keys***
  - Key used for strictly data retrieval purposes.
  - Does not necessarily have a unique outcome.
  - **Narrowing a search** depends on how restrictive it is.
  - Think Chapter's iRewards or hotmail account to look up your iRewards number (the latter = secondary).
  - Typically use a composite key. These types of things.
- Database Keys – Table 3.3 – Overview
- Integrity Rules – Table 3.4 (Entity, Reference Integrity)
  - Most RDBMs enforce integrity rules automatically.
  - Safer to ensure design conforms to entity, referential rules.
  - Other are the NOT NULL, UNIQUE constraints.
    - NOT NULL: ensures every row has a value for that column (i.e. Primary key).
    - UNIQUE: no duplicates exist for that column (i.e. Primary key).
- Relational Algebra
  - Data in relational tables of limited value unless data can be manipulated to generate useful informaiton.
  - Relational algebra manipulates table contents.
  - Eight relational operators.
    - SELECT
    - PROJECT (RESTRICT)
    - JOIN
    - INTERSECT
    - UNION
    - DIFFERENCE
    - PRODUCT
    - DIVIDE
  - Few DBMSs support all of these.
  - Have the property of ***closure*** – that the use of these on existing tables produces new tables without changing the original one.
- SELECT
  - Yields vlaues for all rows found in a table that satisfy a condition.
  - A horizontal subset.
  - “SELECT ALL” vs “SELECT only P_CODE = 311452” (Figure 3.5)
- PROJECT
  - Yields all values for selected attribute(s).
  - A vertical subset (Figure 3.6).
- UNION
  - Combines all rows from two tables, exclude duplicate rows.
  - Tables must have same attribute characteristics (with same domains); i.e. Are union-compatible: same number of columns, compatible data-types and domains.
  - See Figure 3.7
- INTERSECT
  - Only rows appear in both tables (must be union-compatible).
  - See Figure 3.8

- DIFFERENCE
  - All rows not found in another table. Subtraction of one table from another.
  - Must be union-compatible.
  - Not commutative.
  - See Figure 3.9.
- PRODUCT
  - Cartesian product; all possible pairs of rows.
  - See Figure 3.10.
- DIVIDE
  - Uses one single-column table (a) as the divisor and one 2-column table (columns a, b) as the dividend.
  - See slide. Output is a single column with values of a where value in both match.
- Office Hours
  - CS Room 207, moved to 11:00 to 1:00, Tuesdays.
  - Textbook will be veered from in the next class a little bit.
- Next assignment: Next Thursday. (Relational Model, Algebra).
- Review – Relational Database Model
  - Tables (each must have an attribute, combination of them – the primary key that uniquely identifies each row).
  - A key consists of one or more attributes that determines other attributes.
  - Sometimes you can have more than one option for a *primary key* (candidate keys). This is up to the database designer.
- Review – Entity Integrity
  - Requirement: All primary key entires are unique, no part may be null.
  - Purpose: Each row will have a unique identity and foreign key values can properly reference primary key values.
  - Example: *No invoice can have a duplicate number*, nor can it be null.
- Review – Referential Integrity
- Review – Relational Algebra (Operators)
  - The PRODUCT, JOIN, DIVIDE operators can be applied to a pair of tables that are not union-compatible.
  - UNION, INTERSECT, DIFFERENCE require them.
- DIVIDE, cont'd (comparable to integer division in C or Java)
  - “Inverse of product” in a way.
  - Uses one table as the divisor, and one table as the dividend.
  - DIVIDE Example: If milk and eggs are for lunch, which children cannot have any lunch? Allergies (Child, Lunch) DIVIDE Lunch yields Child.
  - DIVIDE is the inverse of PRODUCT and similar to integer divsion: may be contents of the original table that cannot be produced by multiplying the divisor by the quotient.
- JOIN
  - Allows information to be combined from two or more tables.
  - Real power behind relational database; allows use of independent tables (linked by common attributes).
  - Several types.
- Natural JOIN

    - Links tables by selecting only rows with common values in their common attribute(s).
    - Product, Select, Project.
        - PRODUCT of tables is created.
        - SELECT only those rows with common values in common columns.
        - PROJECT is performed to eliminate duplicate columns (common columns).
    - Gives a table without any unmatched pairs and provides only copies of the matches. None of the original tables are affected. Common columns are called ***join columns***.
    - If no match is made, new table match does not include the unmatched row (example, a missing Agent in the Agent table); see slide.
- Equijoin (Theta JOIN)
    - Links tables on basis of an equality condition that compares specified columns.
    - Does not duplicate columns; can do PROJECT as a final step.
    - Condition used to join must be explicitly defined; if a comparison operate other than equality operator is used this is called a ***theta join***. Equijoin is a specific type.
- Inner JOINs
    - A join that only returns matched records.
    - Examples: Natural, Equi, Theta
- Outer JOIN
    - Matched records are returned *and* any unmatched values are returned with null values.
    - Terms left and right refer to the order in which tables are listed in an SQL command.
    - ***Left Outer Join***: See slide. No AGENT_PHONE associated with Smithson, so set to null.
    - ***RIght Outer Join***: See slide. No customers using 333; all set to null in one row.
    - Like an equijoin, does not drop one copy of the common attribute. Examples shown had a PROJECT done.
- ***Data Dictionary***
    - Detailed description of all tables found within user/designer-created database.
    - Contains (at least) all attribute names, characteristics for each table in system.
    - Contains *metadata*, essentially.
    - “The database designer's database” - all decisions that were made.
    - DBMS's internally stored data dictionary contains additional information about relationship types, etc.
    - See Table 3.6; slide.
- ***System Catalogue***
    - Like Dictionary; contains metadata.
    - Data dictionary is a subset of this. Contains thins like creator, access privileges, etc.
    - Can be queried just like any user-created table.
- Homonyms
    - Similar sounding with different meaning, or identically spelled.
    - Board, bored.
    - In databases, use of the same attribute name to label different attributes.
        - C_NAME: Customer, Client, Consultant, etc.
        - Avoid the use of them; db dictionary is useful for this.
- Synonyms
    - Different words with same meanings.
    - In database, indicates use of different names to describe same attribute.

    - Avoid them!
- Relationships in RDB
    - 1:M is relational modeling ideal.
    - 1:1 should be rare.
    - M:N cannot be implemented but can be changed to 1:M.
- See paper.
- Relational Algebra
    - Not in the textbook; only have to know what we learn here today.
    - Recall the relational operators from last day.
        - SELECT, PROJECT: Unary operartors.
        - INTERSECT, UNION, DIFFERENCE, PRODUCT, DIVIDE: Binary operators.
        - JOIN: Two or more.
    - Indexes
        - Orderly arrangement to logically access rows in a table.
        - Can be used to retrieve data more efficiently.
        - Can also used by the DBMS system to retrieve data ordered by specific attributes (i.e. By last name).
        - An ordered arrangement of keys and a set of pointers.
        - Unique index is an index in which the index keys can have only one pointer value (row) associated with it. Creates a unique index on the primary key column(s) by the DBMS.
        - Each index is associated with only one table.
    - Logic operators.
        - ^ AND operator: if and only if p and q are true.
        - v OR operator: if at least one of p and q are true, false only when both are false.
        - Negation: NOT.
    - Quantifiers
        - Universal (“for all”) - if and only if is true for all elements.
        - Existential (“there exists”) - true for at least one.
    - Relational Algebra proposed as basis for query languages in 1970.
        - Five operations at the time; see slide.
        - Other operations can be defined in terms of these (such as what we've seen).
        - High-level procedural language consisting of set of operators.
        - Work on one or more relations (tables). Both operands and results are tables.
        - See notation.
    - Selection Notation
        - Sigma_condition(R) where condition is a predicate that can include logic operators and comparison operators (<,>,etc.).
        - See slides.
        - Example: Staff.
            - Have a Staff relation with attributes Staff Num, FName, LName, Dept, Salary.
            - Selection of those with salary > 150000 would give us a new table with two tuples/ rows matching this.
    - Projection Notation
        - PiAttributes(R)
        - Example: Staff.

        - Gives a new table with two rows and only 3 of the attributes: FName, LName, and Salary.
        - Doing the operation on the result of the selection.
- Join Notation
    - Natural join: see slide.
    - Theta join: same where C is an arbitrary condition. See slide.
        - For equijoin, result with a combined table with say all those who are both students and staff.
        - Contains both sets of attributes.
    - Left, Right Outer Join has a simliar symbol with one side open. See slide.
    - Among the most difficult operations to implement efficiently; one of the reasons why relational databases tend to be slow.
- Relational Calculus
    - Provides a declarative way to specify queries; specifies what is to be retrieved, now how to retrieve. Difference to procedural language.
    - Predicate calculus. Predicate: sentence with finite number variables which becomes a statement when specific values are substituted.
    - Tuple Relational Calculus.
        - Tuple variables: whose only permitted values are tuples from the relation.
        - If F is a formula, then to find the set of tuples (S) for which F(S) is true, we state {S| F(S)}
        - Queries are formulas, defining sets using:
            - Constants
            - Predicates
            - Logical operators
            - Quantifiers
        - Bound, Free Variables
            - Free variables take on values of tuples.
            - Bound variables are those bound by quantifiers; variable that is not bound is free.
        - Building a formula.
            - Should be unambiguous.
            - Made out of “atoms”
                - R(T) where T is a tuple variable, R is a relation.
                - T.a theta W.b where a,b attributes of T and W, and theta is one of (<,>,<=,>=,=,!=).
                - T.a theta c or c theta T.a where c is a constant.
            - Can be built recursively from atoms.
                - An atom is a formula.
                - If F1, F2 are both formulas, conjuncion, disjunction, negation are also formulas.
                - If F is a formula with free variable X, see slide.
            - Example
                - Find all tuples... (see slide)
                - Bring back only last names.
                - Using existential operator, so considered bound.

- Selection Example
  - In relational algebra: Sigma_CONDITION(R)
  - In tuple relational calculus, we write: {S | R(S) ^ CONDITION(T)}
  - Tuple variable is free (in example).
- Projection Example
  - In tuple relational calculus we write: {S.A1,S.A2,... | R(S)}
  - Tuple variable is free (in example).
- Safe Queries
  - Some queries can generate an infinite set (example: {S|not(STUDENT(S))} – anything that is not a student).
  - Way to make a query safe is to restrict the domain. Each query expressed in relational algebra can be expressed as a safe query in relational calculus; therefore, equally expressive.
- Relational Completeness
  - Query language is at least expressive as relational algebra is said to be relationally complete.
  - Relational algebra, Safe relational calculus, SQL.
- Domain Relational Calculus
  - Similar to tuple RC, but ranges over domain (attribute) values rather than tuples (columns, rather than rows).
  - Domain relational calculus is equivalent to tuple relational calculus when both are safe.

- Assignment #1 Solutions
  - True/False
    - 1 – False (actually definition of independence)
    - 2 – False
    - 3 – False (relational model is implementation standard)
    - 4 – False (the other way around)
    - 5 – True
    - 6 – True
    - 7 – False (recall: black box)
    - **8 – False** (software dependent)
    - 9 – True
    - **10 – False** (hardware-independent as well)
  - Fill in the blanks.
    - Hierarchical
    - Metadata
    - Logical
    - **Context**
    - Entity Relationship
    - Data Redundancy
    - Constraint
    - Entity
    - Attribute
    - Business Rule
  - Short Answers

- KOM: Deletion anomaly.
- Conceptual model: Decent description (key: “global view”).
- Disadvantages of DB systems: Increased costs, management complexity, maintaining currency, ...
- Business Rules

*Entity Relationship Modeling* (Chapter 4)

- Assignment #2 posted.
- Review of last time:
  - ^ AND, v OR, ! NOT, quantifiers.
  - Notation
  - Relational Algebra (High-Level Procedural) vs. Relational Calculus (Declarative – what you want, not how you want it).
- **SQL** is more expressive than either; includes aggregate functions and constructs (such as GROUP BY, HAVING).
- Extensions to relational algebra, relational calculus have been proposed to overcome this disparity. Not become standard yet.
- Entity Relationship Diagram represents the conceptual database as viewed by the end user; not how the DBMS sees the data, how stored and accessed, etc.
- Depict the main components of the database: entities, attributes, relationships.
- Entities: Real-world objects; entity refers to entity set (table), not to a single occurence (a row).
- Attributes: Characteristics; listed inside the rectangle in Crow's Foot Notation. Usually use capital letters; have some sort of beginning of attribute relating to table name (e.g. STU_LNAME for table STUDENT).
- ***Required Attributes*** – must have a value (cannot be left empty); optional attributes.
- Domains – All attributes have a domain; set of possible values. May share.
- ***Identifiers*** (Primary Keys): mapped to primary keys (PKs) in tables. Underlined in diagram.
  - Ideally, one attribute.
  - Sometimes, need to use composite keys. Underline both.
- ***Composite Attributes***: can be subdivided to yield additional attributes vs. Simple attributes.
  - To facilitate detailed queries, best to break these down.
  - For example, 1 attribute for an address vs. Postal code, etc.
- ***Single-Valued Attributes***: Not necessarily a simple attribute – merely one that has a single value. For example, serial number.
- ***Multi-Valued Attributes***: Have many values. Person could have many phone numbers. Crow's Foot Notation does not identify multi-valued attributes. Chen Notation is a bit more descriptive in this regard.
  - Should not be implemented in the relational database.
  - Each column/row intersection in a relational table should represent a single data value.
  - Solution: Create new attributes or an entirely new entity.
    - For example: CAR_TOPCOLOR, CAR_BODYCOLOR, etc. Instead of just CAR_COLOR. Not the best solution; can create structural problems within the table, and could result in many null values.

        - Better: New entity composed of original attribute's components: CAR_COLOR entity. 1-to-many relationship to original.
- ***Derived Attributes***: Value need not be stored in database; can be calculated (derived) using an algorithm. Example: Total cost of an order.
    - Crow's Foot does not identify them from others.
    - Advantage of having actually stored: saves CPU processing time, data access time. Readily available. Can be used to keep track of historical data.
    - Disadvantage: Requires constant maintenance to ensure current, especially if any values used in calculation change.
- Relationships: Association between entities. Named by a verb (active or passive).
    - Always operate in both directions (why 2 business rules).
    - When written down, easy to see 1:M relationship (see slide).
- Connectivity, ***Cardinality***
    - Connectivity: Classification of relationship (1:M, etc.).
    - ***Cardinality***: Minimum, maximum number of occurrences associated with one occurence of a related entity. Established using business rules, as well. Indicated by placing (min,max) beside entities (Figure 4.7).
- Existence Dependence
    - An entity is said to be this if can exist only when it is associated with another related entity occurence. For example, EMPLOYEE claims DEPENDENT; this latter entity is existence-dependent on EMPLOYEE entity.
    - Independence: Reverse is true, as well. Referred to as strong, regular entity.
- Relationship Strength
    - Based on PK. How defined.
    - Primary key of one entity appears as a FK in a related entity.
    - Weak (Non-Identifying) Relationship: PK of related entity does not contain a primary key component of its parent entity. See slide. Dashed line.
    - Strong (Identifying) Relationship: Contains a component. Solid line. Order in which tables are created is important; load the “1” side of a 1:M first.
- ***Weak Entity***: Existent-dependent (on parent), has a primary key partially or totally derived from parent entity in a relationship.
- Relationship Participation: For each entity, participation in a relationship is either optional or mandatory.
    - Optional: drawing a small circle on the side of the optional entity in Crow's Foot. Minimal Cardinality would be 0.
    - Mandatory: Minimal has to be 1.
    - Not the same as relationship strength – the latter depends on how primary key is formulated whereas the other is based on business rules.
    - See Figure 4.12, 4.13, 4.14.
- See Table 4.3 for Crow's Foot Notation with Cardinalities.
- Relationship Degree: Number of entities associated with a relationship.
    - Rare, but happens: recursive (unary relationship) – one employee manages other employees.
    - Binary relationships: two in a relationship; most common.
    - Most higher-order are decomposed into equivalent binary relationships.
    - Ternary and higher: associated among three+ different entities. For example: doctor, patient,

drug.

- ***Recursive Relationship***: A relationship can exist between occurences of the same entity set. Such a condition is found within a unary relationship.
- Associative (Composite) Entities: Decompose M:N relationships into 1:M relationships.
  - A bridge or composite entity.
  - Composed of primary keys of each of the entities to be connected.
  - May also contain additional attributes that play no role in the connective process.
  - See Figure 4.24, 4.25.
- Developing an ***ER Diagram***
  - Process of DB Design is an *iterative* process that usually includes:
    - Creating a detailed narrative of the organization's description of operations.
    - Identifying the business rules based on the description of operations.
  - Identifying the main entities, relationships from the business rules.
  - Develop the initial ER diagram (very simple to begin with).
  - Identify attributes, primary keys that adequately describe entities.
  - Revise, review.
    - During the review process, likely will come up with additional objects, attributes, relationships.
    - Each round of reviews might yield more components, clarifications.
  - Even when it comes down to the end, might not be the perfect representation, but learned a lot.
  - Process is repeated until end users, designers agree it is a fair representation; or run out of time, money.
  - During the design process, designer relies not only on interviews, also on examination of business forms, reports used in daily operations of the business.
- Database Design Challenges: *Conflicting Goals*
  - Designers often make design compromises that are triggered by conflicts goals such as:
    - adherence to design standards (elegance)
    - processing speed
    - information requirements
  - May be a conflict there; best design from designer might not give company what they need. Database design should conform to design standards (minimize redunancy, nulls).
  - Standards allow you to work with well-defined components and to evaluate interaction of these components with some precision.
- Processing Speed
  - Many organizations (with a lot of transactions) may prefer processing speed.
  - Means minimal access time, minimize number and complexity of relationships.
  - Ideal design may have a 1:1 relationship to avoid nulls while higher processing speed requires combining two tables using dummy entries to avoid nulls.
- Information Requirements
  - Timely information.
  - Complex requirements may expand number of entities, attributes (composite to simple attributes).
  - Clean structures, transaction speed may be sacrificed to ensure maximum information gathering.

- It's all about **compromise**. Part of the design process.
- The Design Process
  - Completed diagram is actual blueprint of the database.
  - Composition must reflect an organization's operations *accurately* if database is to meet data requirements of organization.
  - Forms basis for a final check on whether included right entities, attributes, right relationships. Also used as a final crosscheck against proposed data dictionary entries.
  - Can also let you communicate more precisely with those who comissioned desgn.
  - Finally, serves as implementation guide to those who create it.
- ER Modeling
  - Even the best process produces an ER diagram that requires further changes.
  - Despite this, is essential in the development of sound design.
  - Using ERDs yields thorough understanding of how an organization really functions.
- Document, Document, Document
  - Put all design activities in *writing*.
  - Review what you have written.
  - Documentation helps design process *stay on track*, allows for easy modification and understanding in the future.
- Tiny College Example
  - Realize that when a company hands information, not all details are applicable.
  - Interview Information 1: Divided into several schools.
    - Each school administered by a dean (who is a professor).
    - Each professor can be the dean of only one school.
    - A professor is not required to be dean of any school.
  - What does this tell us? Entities: SCHOOL, PROFESSOR.
    - 1:1 relationship (is dean of) between PROFESSOR and SCHOOL.
    - Optional (cardinality 0,1) since professor does not have to be a dean in direction of PROFESSOR to SCHOOL.
  - 1:1 Relationships often indicates a misidentification of attributes as entities.
    - Could be eliminated by storing the dean's attributes in the SCHOOL entity.
    - Would make it easy to answer questions such as “Who is the dean?”
    - Downside: Duplication of data (could lead to data anomolies).
    - In this case, choice is to retain the 1:1 relationship. Choices like this *should be defensible*.
    - ERD Segment (see slide) – weak relationship because of dotted line. Optional because of the O.
  - Interview Information 2: Each school has several departments.
    - Smallest number is 1, but largest number is indeterminate (cardinality: N).
    - Each department belongs to only a single school.
  - Need another entity: DEPARTMENT.
    - Has one or more department per school, but each belongs to one school (1:M relationship) - “operates” between SCHOOL and DEPARTMENT (name of relationship will vary between designers).
    - See ERD Segment slide.
  - Interview Information 3: Courses

        - Each department may (but not necessarily) offer courses.
        - Each course is offered by only one department.
    - New entity: COURSE.
        - 1:M relationship (“offers”) between DEPARTMENT, COURSE.
        - COURSE entity is optional to DEPARTMENT.
    - Interview Information 4: Class is a section of a course.
        - Department may offer several sections of same course.
        - Each section of a course is taunt by a professor (given time and place).
        - Course may exist in course catalogue even when not offered.
    - New entity: CLASS
        - 1:M relationship (“generates”) between COURSE, CLASS.
        - Existence of a COURSE does not guarantee existence of CLASS, CLASS is optional to COURSE (CLASS is dependent on COURSE).
    - Interview Information 5: Each department employs one or more departments.
        - Each professor is employed by only one department.
        - One and only one of the professors employed by a department chairs it.
        - No professor is required to chair one.
    - This tells us...
        - 1:M relationship (“employs”) between DEPARTMENT, PROFESSOR.
        - 1:1 between DEPARTMENT and PROFESSOR (“chairs”).
        - PROFESSOR is not required to accept position as chair, DEPARTMENT is optional to PROFESSOR.
        - See slide.
    - Interview Information 6: See slide.
        - 1:M relationship between PROFESSOR and CLASS (“teaches”).
        - Optional.
    - Interview Information 7: Students
        - Student may enroll in several classes.
        - Each student may enroll in up to 6.
        - Each class may have up to 35.
        - A class can exist even though no students have enrolled in it (yet).
    - Tells us...
        - M:N relationship (“enrolls”) between STUDENT, CLASS (create bridging entity ENROLL).
        - If a class exists, class does not occur in ENROLL, therefore ENROLL is optional to STUDENT.
        - See slide.
    - Interview Information 8: Majors (see slide).
        - 1:M relationship (“has”) between STUDENT, DEPARTMENT.
        - DEPARTMENT can exist without students, and a STUDENT is not necessarily associated with a DEPARTMENT.
    - Interview Information 9: Advisors (see slides).
    - Interview Information 10: Rooms (see slides).
    - **Table 4.4**: Components of this ERM (all information built up, relationships, connectivity).
        - See final diagram (slide).

        - Piece-by-piece, very simple. When all put together, looks complex.
        - Eventually, have to start clustering to make a more readable diagram because too many entities.
- Video Rental Store Example (see slides).
    - The idea of having more than item in a single transaction: a LINE.
    - More than one LINE on an invoice.
- Review: Relationship Strength
    - A weak one exists if the primary key of related entity does not contain a primary key component of its parent entity (otherwise, is a strong relationship).
    - Alternatively, can refer to these relationships as non-identifying (weak) and identifying (strong).
    - A strong, identifying relationship ensures dependent entity can only exist when it is related to the parent entity. Does not mean a weak relationship *is bad*.
- Review: Weak Entity
    - Meets two conditions:
        - Existent dependent.
        - Primary key partially or totally derived from parent entity.
    - We find weak entities in *strong relationships*. Cannot exist at the same time.
- Review: Developing an ER Diagram – remember: Iterative Process.
    - A completed ER diagram is the actual blueprint of the database.
    - Serves as the implementation guide to those who create the actual database.

*Advanced Data Modeling* (Chapter 5)

- Assignment #3 posted.
- Extended Entity Relationship Model
    - Result of adding more semantic constructs to the original ER model.
    - A diagram using this model is called an extended entity relationship diagram (EERD).
- Entity Supertypes, Subtypes
    - An ***entity supertype*** is a generic entity type related to one or more ***entity subtypes***. (Parent and children).
    - The entity supertype contains common characteristics, entity subtypes contain unique characteristics of each entity subtype.
    - Two criteria to help the designer determine when to use them:
        - There must be different, identifiable kinds or types of the entity.
        - The different types of instances should have one or more attributes that are unique to that type of instance.
    - For example, a EMPLOYEE supertype with subtypes for specialized ones such as PILOT, ACCOUNTANT, ...
    - Example: The attribute in the supertype tells what type of subtype each instance belongs to. Not doing this creates a great deal of nulls because of unique attributes. (See slides).
- Null Reminder
    - Create problems because they have *many* different meanings. An unknown attribute value, a knwn but missing one, a N/A condition.
    - When functions like COUNT,SUM,AVERAGE are used, there can also be problems.

- Creates problems when relational tables are linked.

- ***Specialization Hierarchy***
  - Entity subtypes, supertypes are organized in a specialization hierarchy.
  - Depiction of arrangement of higher-level supertypes and lower-level subtypes.
  - Sometimes refered to as "is-a" relationships. A pilot *is an* employee.
  - Subtype can only exist within context of supertype (and can only have one supertype to which it is directly related).
  - See notation in slide (Figure 5.4). Category symbol (circle with two lines underneath it).
    - **O** means overlapping (can be both).
    - **D** means distinct, disjoint (only one or the other).
  - Provides means to:
    - Support attribute ***inheritance***.
    - Define special subtype attribute (***subtype discriminator***).
    - Define disjoint/overlapping constaints (and complete/partial constraints).
- Inheritance
  - Property of inheritance enables entity subtype to inherit attributes, relationships of subtype.
  - Inherit primary key.
  - Data is not repeated (simply have the link).
- Subtype Discrimination
  - Attribute in supertype entity that determines to which subtpe the supertype occurence is related.
  - Default comparison condition for discriminator attribute is the equality comparison.
  - There may be situations when not based on this (i.e. Greater than).
- Disjoint, Overlapping Constraints
  - Disjoint subtypes (non-overlapping) contain a unique subset of supertype entity set.
  - Overlapping ones contain non-unique subsets – each entity instance of the supertype may appear in more than one subtype.
  - Table 5.1: If implementing overlapping subtypes, need to specify whether he is or is not each of the subtypes (single discriminator with a set of these logical values).
- ***Completeness Constraint***
  - Specifies whether or not each supertype occurence must also be a member of at least one subtype.
  - Can be partial or total:
    - Partial: Not every supertype is a member of a subtype. (Circle over a *single* line).
    - Total: Must be a member of at least one subtype. (Circle over two lines).
  - See Table 5.2
- ***Specialization*** and ***Generalization***
  - Specialization is the top-down process of identifying lower-level subtypes.
  - Generalization is the bottom-up process of identifying higher-level supertypes. Based on grouping common characteristics, relationships.
- Figure 5.2: Specialization Hierarchy in detail.
- Entity Clustering
  - In an initial entity relationship diagram, typically only a few entities.
  - However, as diagram is developed, can be hundreds of entities, relationships.
  - As the design approaches copmletion, can be so many entities, relationships diagram

becomes crowded, unreadable.
    - Considered virtual; just helps clean diagram. Not going to have them in the very final blueprint.
    - Temporary entity used to represent multiple entities, relationships in order to simplify the diagram (make it more readable). When using this, attributes of combined entities are *not* displayed.
    - See Figure 5.5 (for Tiny College).
- Selecting Primary Keys
    - Most important characteristic of an entity is its primary key.
    - Function of it is to guarantee entity integrity.
    - Primary keys, foreign keys work together to implement relationships in the relational model.
    - ***Natural key*** (natural identifier): real-world identifier used to distinguish real-world objecs (credit card acct numbers). If an entity has one of these, usually best choice for primary key.
    - Desirable characteristics:
        - Unique (see slide).
        - Non-intelligent.
            - Should not have a semantic meaning other than to uniquely identify each entity instance.
            - For example, student number would be preferred over combinaiton of last name, first name.
        - No change over time.
            - Should be permanent, unchangeable.
            - If an attribute has semantic meaning, might be subject to update.
        - Preferably single-attribute.
            - Desirable, not required.
            - Composite primary keys can cause primary keys of related entities to grow, making coding more cumbersome.
        - Preferably numeric.
            - Can use internal routines, etc.
            - Better managed when numeric.
        - Security-compliant.
            - Selected primary key must not be composed of any attributes might be considered a security risk.
            - For example, using SIN is not a good idea.
- When to use Composite Primary Keys
    - Two cases where useful:
    - As identifiers of composite entities.
        - Created to break an M:N relationship into two 1:M relationships.
        - Ensures cannot be duplicate values.
    - As identifiers of weak entities.
        - A weak entity in a strong, identifying relationship with a parent entity is normally used to represent one of two situations:
        - Real-world object that is existence-dependent on another one.
        - Real-world object represented in as two separate entities in a strong, identifying relationship.

        - For example, real-world invoice represented by INVOICE, LINE.
        - LINE does not exist in real world, but rather as part of an INVOICE.
- When to use Surrogate Primary Keys
    - ***Surrogate Primary Keys***: When no primary key exists in the real world, or when existing natural key might not be suitable. For example, an invoice number.
    - No meaning in user's environment – exists only to distinguish one entity from another.
    - One practical advantage is that values for it can be generated by DBMS.
- Implementing 1:1 Relationships
    - Basic rule is: put primary key of the "one" side on the "many" side as a foreign key, but how does this apply to a 1:1 case?
    - Two options: Put foreign key in both or one entity.
    - Putting it in one avoids duplication, so it is considered best. But which one?
        - If one is mandatory (and other optional), place key in the entity on the optional side and make it mandatory.
        - If both optional, select that which causes fewest null values.
        - If both mandatory, select that which causes fewest null values (or consider revisting model – maybe they shoudl be one entity).
- Maintaining a History of ***Time-Variant Data***
    - Data stored reflect not only current data but also historic data.
    - Normally, when data is changed, overwriting it.
    - There are situations in which history must be preserved.
    - Time-variant data refers to data whose values change over time and which you **must** keep a history of the data changes.
    - Keeping the history is equivalent of having a multi-valued attribute. Therefore, create a new entity with a 1:M relationship with original entity.
    - See Figure 5.8.
- ***Design Traps***
    - Creating a data model requires proper identification of data relationships among entities.
    - Sometimes not identified correctly.
    - Design trap occurs when relationship is improperly, incompletely identified.
    - ***Fan Trap***: When one entity in two 1:M relationships with other entities.
- ***Redundant Relationships***: Multiple relationship paths between related entities.
    - Some designs use them to simplify design.
    - Do not have to exist.

*Normalization of Database Tables* (Chapter 6)

- Normalization is the process of evaluating, correcting table structures to *minimize data redundancies*. Reduces likelihood of data anomalies.
- This normalization process involves assigning attributes (to tables) based on concept of determination.
- Creating new tables to solve problems.
- Remember: *Determination*. A determines B indicates knowing value of attribute A means value of B can be looked up. A -> B.
- Normalization works through series of stages known as normal forms. First three stages:

    - First Normal (1NF)
    - Second Normal (2NF)
    - Third Normal (3NF)
- Each worse than the next. For most purposes, 3rd is as high as needed in normalization process. Properly designed 3NF structures also meet requirements for 4NF.
- As you go higher in normal forms, creating more tables, but also slowing down performance of database. Like anything else, is a tradeoff. More JOIN operations required to get information.
- Sometimes part of database design are denormalized in order to meet performance requirements. Gives a lower normal form. Trade-off for increased performance is more redundancy.
- ***Prime Attribute***: Any attribute that is at least part of a key (primary, candidate – minimal superkey – potentially a good primary key but was not chosen for a reason).
- ***Nonprime Attribute***: Not part of any candidate key.
- For the purposes of this section, assumption is only one candidate key (more = more complex).
- Two common situations in which we use normalization:
    - After initial design, normalization can be used to analyze relationships that exist among attributes within each entity to determine whether normalization could improve structure.
    - When modifying existing data structures (flat files, spread sheets, ...), database designer can use normalization process to improve existing structure to create an appropriate design.
- Need for Normalization
    - Suppose we have a small construction company that manages several building projects.
    - Each project has its own project number, name, employees assigned to it.
    - Employee: number, name, classification.
    - Company charges its clients by billing hours spent on each contract.
    - See slides. Table 6.1: Type of report they want to generate.
    - Easiest short-term way to generate report would be a tbale whose contacts correspond to reporting requirements. Unfortunately, some structure deficiencies to this approach. See Figure 6.1.
        - Null spaces since using project number for a primary key but only entered once for each project.
        - Repetition of employee information. Job classes instead of some sort of symbol for them. Typed in over and over again. Update anomalies possible (modified in multiple places).
        - Shoud tell you that you'll probably get typographic errors.
        - Insertion Anomalies: If new employee is inserted and not assigned to a project yet, must create a “phantom project” in order to enter data.
        - Deletion Anomalies: if only one employee to a project, project information would be deleted as well.
- Objective of ***normalization*** is to ensure each table conforms to concept of well-formed relations. ***Well-formed tables*** should have characteristics:
    - Each table represents a single object.
    - No data item will be unnecessarily stored in more than one table.
    - All nonprime attributes in a table *dependent* on primary key: entire primary key and nothnig but the primary key.
    - Each table is void of insertion, update, deletion anomalies.

- ***Normal Forms***
  - 1NF: Table format, no repeating groups, primary key identified (any relational table).
  - 2NF: 1NF + no partial dependencies.
  - 3NF: 2NF + no transitive dependencies.
  - Boyce-Codd NF: Every determinant is a candidate key (special 3NF cases).
  - 4NF: 3NF + no independent multi-valued dependencies.
  - Higher forms not likely to be encountered in business environment.
- Candidate Keys: Recall – minimal, non-reducible superkey. We'll initially assume for every table there is only one candidiate key and it is the primary key. Will be the case in this class in general.
- Functional Dependence: Normalization process works one relation at a time, identifying dependencies and normalizing the relation.
  - Identify dependencies and break up relation into new relations based on identified dependencies.
  - Breaking a table into multiple ones.
  - Attribute B is ***fully functionaly dependent*** on A if each value of A determines one and only one value of B. Attribute A determines B if all rows in the table that agree in value for A also agree in value for B.
  - If attribute B is functionally dependent on composite key A, but not on any subset, B is fully functionally dependent on A.
  - Partial Dependence: Determinant is only part of the primary key (assuming one candidate key). If (A,B) -> (C,D),B -> C, and (A,B) is primary key, functional dependence B -> C is example of partial dependence.
  - Transitive Dependency: functional dependencies such as X -> Y, Y -> Z (signalling dependency, willy be referred to as the transitive dependency) and X is primary key. X -> Z is a transitive dependency because X determines value of Z via Y.
- Conversion to 1NF: Repeating group is a group of multiple entries of same type in a relation that exist for any single key attribute occurence.
  - Relational table must not contain any repeating groups.
  - If they do exist, must be eliminated by making sure each row defines a single entity instance.
  - For example, filling in nulls in PROJ_NUM. See Figure 6.2.
  - Three-step procedure: Eliminate repeating group (eliminate nulls by making sure each repeating group contains an appropriate data value), identifying primary key (PROJ_NUM + EMP_NUM), identify all dependencies.
    - See slides.
    - See all dependencies. Obvious dependency: that of the primary key. Dependency diagram.
    - Dependency Diagram: Primary key attributes bold, underlined, different color. Arrows above attributes indicate all desirable dependencies (primary keys), arrows below all less desirable dependencies (partial, transitive). See Figure 6.3.
- Conversion to 2NF: Partial dependencies should be used with caution; table with them is still subject to data redundancies (data anomalies).
  - Only when first normal form has a composite primary key. If it has a single-value one, already in 2NF.

- Make new tables to eliminate partial dependencies. It is important the component remain in the original table as well (act as foreign keys for new relationship).
- In our example: three tables come out: PROJECT, EMPLOYEE, ASSIGNMENT.
- Still transitive dependency (can generate anomalies) but tables are in 2NF. Partial dependency only exists when table's primary key have several attributes. See Figure 6.4.

- Conversion to 3NF: Transitive dependencies have to be removed.
  - Make new tables, reassign dependent attributes.
  - Similar to previous conversion.
  - For every transitive dependency, write a copy of its determine as a primary key for a new table. See slides for example.
  - Determined values removed.
  - We get the following in our example: PROJECT, EMPLOYEE (EMP_NUM,EMP_NAME,JOB_CLASS), ASSIGNMENT, JOB (JOB_CLASS, CHG_HOUR). See Figure 6.5.
  - The primary key no longer relies on 1+ non-prime attributes to functionally determine other non-prime attributes.
  - It is very important to achieve 2NF before moving on to 3NF. Partial dependencies must be resolved before transitive dependencies.
- Multiple Candidate Keys
  - If a table has more than one, and one of them is a coposite key, can be more partial dependencies, etc.
  - Those dependencies can be perceived as transitive dependencies. See slide.
- Higher-Level Normal Forms
  - Occasions when higher normal forms are useful, such as Boyce-Codd and 4NF.
  - Boyce-Codd is just a special case.
    - Candidate key is a minimal superkey. When a table contains only one candidate key, 3NF and BCNF are equivalent.
    - Moat tables conform to BCNF when 3NF is reached.
    - A table can be in 3NF and fail to meet if it has a nonkey attribute that is a determinate of a part of the primary key. See Figure 6.7.
      - To convert this, first change primary key to A+C (where C -> B and C is nonkey). Depedency C -> B implies C is a superset of B.
      - Table beecomes 1NF.
      - Proceed as usual.

Reminders: TA not available in Learning Center this Friday.
Assignment 3 due on Thursday.
Tuesday, Feb 14th will be a review class.
Midterm is Thursday, Feb 16th in-class.

Midterm Format: Very similar to Assignments.

Assignment #3 Solutions:

Fill in the Blanks

Domain
Non-Overlapping
Composite (Know this difference)
Entity Instance or Row
Recursive
Time-Variant Data
Derived
Generalization

Short Answers
Business Rules

Topics on Midterm:

**Chapter 1**
Data vs. Information
What is a database?
DBMS – Advantages, Disadvantages
Data Inconsistency – could even be spelling mistakes.
Formats of Data (Logical, Physical)
Data Redundancy
Structural Dependence
Data Dependence/Independence – Know what they mean.

**Chapter 2**
Data models – entities, attributes, relationships, constraints.
Relationships – three types (1:M, M:N – broken down, 1:1 – tends to be rare)
Constraints – help ensure data integrity.
Business Rules
Network Model, Hierarchical Model
Object-oriented Data Model (Semantic, adds more meaning) – objects contain data and relationships rather than just a simple entity.
Entity sets, instances.
Connectivity (classification – 1:M, M:N, 1:1)
Degrees of Data Abstraction (External, Conceptual, Internal, Physical)
Relational Model (Codd, Conceptual Simplicity – allows data to be viewed logically rather than physically)

**Chapter 3**
Tables = Relations
Primary Key (Surrogate Keys), Foreign Keys ensure integrity of data.
Data Types (Numeric, Character, Logical, Date)
Determination
Superkeys, Candidate Keys
Entity Integrity (Purpose, Example)
Data Redundancy (Controlled Redundancy make RDBs work)

Foreign Key, Referential Integrity
System Catalogue – Detailed system data dictionary; describes all objects.
(Designer makes data dictionary, software makes system catalogue).
Homonyms, Synonyms
Splitting M:N Relationships

**Relational Algebra**

Relational Algebra
Closure
SELECT, PROJECT, PRODUCT
Union-Compatible: Important. Share same number of columns, corresponding columns have compatible data types, domains. (UNION, INTERSECT, DIFFERENCE require pair of tables to be union-compatible).
UNION, INTERSECT, DIFFERENCE
JOIN – real power behind RDBs, allow use of independent tables linked by common attributes, but is also one of the most expensive.
Types of JOIN – know steps, final outcome.
Right Outer Join, Left Outer Join: Remember, does not drop one copy.
Notation
Relational Algebra vs. Relational Calculus
Algebra: High level procedural language.
Calculus: Declarative way – not how to retrieve it. (No queries on midterm). Equally as expressive.
Relationally Complete
Safe Queries

**Chapter 4**

Entity Relationship Diagrams
Composite, Simple Attributes (cannot be broken down further)
Single-valued, Multi-valued Attributes (think car color)
Derived (Computed) Attributes
Identifier = Key, primary keys are underlined in ERD.
Relationship: Association between entities. Goes in both directions.
Cardinality: Min, max number of occurences associated with one occurence of a related entity
Unary Relationship (Recursive Relationships)
Binary Relationships
Higher Order Relationships should be decomposed to Binary
Existence-Dependence/Independence
Design Compromises
Completed ERD actual blueprint of database.

**Chapter 5**

Subtypes
Inheritance
Specialization Hierarchies (know the diagram)
Partial, Total Completeness
Specialization, Generalization

**Chapter 6**

Normalization

Denormalization