Intro to Database

What is a Database Management System?

1. System software for creating and managing databases. It provides users and programmers with a systematic way to create, retrieve, update and manage data (manages the data, database engine, and database schema)
2. Manages
    a. Data
    b. Database Engine
        i. It allows data to be accessed, locked, and modified
    c. Schema
        i. Defines the database's logical structure.
    d. **These three elements provide concurrency, security , data integrity and uniform administration procedures

Why are DBs important?

It serves as an interface between the database and end users/application programs, ensuring that data is consistently organized and remains easily accessible

What makes a database a database?

1. Built in organizational structure
2. All the three things listed above under manages

Why do we need DBMS's? Why not just use files?

o Redundancy and Inconsistency

1. Want to reduce redundancy because it makes queries run faster if there are less redundancy
2. Distributed DB models
    a. Run no SQL and are javascript based DBs
        i. Like Redundancy because it is easy to update and having multiple copies of updates in multiple serves
        ii. Ex. Facebook

o Multiple file formats, duplication of information in different files

1. For MySQL database engine there are two different file formats
    a. Everything together file format
        i. Easy to cross reference data
    b. And breaking every db into 3 files
        i. Faster

o Accessing data from different applications – new program for each task

o Data Isolation – multiple files and formats

o Data integrity (e.g. bank balance) – becomes hard code – hard to change or add

- Foreign keys
- advanced sql commands like checks and of the sort

-

o Atomicity of updates – failures leave info in state of flux – fund transfer example
o Concurrent Access by multiple users – needed for performance – unstable data if two read at same time
o Security
o Data Abstraction – levels of abs: Physical, Logical, View
o Physical Data Independence

1. Dealing with a model of the data instead of the low level data itself
2. **You can switch implementations. You can theoretically switch out the database from underneath it and most things should still work.

Relational Model

Schema vs. Instance

- Schema: the logical design of the database; doesn't and shouldn't change (static)
- Instance: A "snapshot" of the data stored in the database at a given time; should and often does change

Models – represent things in the real world – at extreme accuracy, a model is a clone - How much detail?

A relation is a table with columns/attributes and rows/tuples

Superkey vs. Candidate Key vs. Primary Key

- Superkey
  - Set of one or more attributes that, taken collectively, allows us to identify uniquely an entity in the entity set
  - Combination of columns (attributes) that uniquely identifies any row
  - Not a candidate key nor a primary key
- Candidate key
  - Minimal super key - super key for which no proper subset is a super key
  - Minimal set of attributes (columns) necessary to identify a tuple (row)
- Primary Key
  - Is a candidate key that is chosen by the DB admin/designer as the principal means of identifying a tuple within a table

Primary Keys - What is a primary key and what makes a good primary key?

o Doesn't change, Unique. Non-null / Everyone has one, Meaningful in some way

Relations vs. Tables

- Relation: Relational Algebra + theoretical mathematical DBs, has no order, does not have duplicates, and is a set
- Table: SQL + actual DB, has order, can have duplicates, and is a list

DDL vs. DML

- DDL: Data Definition Language; effects schema using create table and alter table

- DML: Data Manipulation Language; effects instance using select, insert, update, and delete

Sailors and Bank Examples – Relational Algebra (RA)

Relational Algebra operators:

Select: $\sigma_p(r)$ (p = selection predicate, r = relation(table)) comparisons in the selection predicate (=, <, <=, >, >=, !=), can use connectives(AND, OR, NOT), limits the rows based on what we're looking for

Project: $\pi_{A1, A2, \ldots, Am}$ (r)((A_i = attribute) return specific attributes (columns) from a relation

Select & Project

Union: r U s (r and s are two relations) unify (combine) tuples from two relations, r and s must have the same degree, the corresponding attribute domains must have a compatible type

Difference: r − s (where r and s are two relations) find the tuples that are in one relation but are not in another, r and s must have the same degree (same number of attributes), the corresponding attribute domains must have a compatible type

Cross (Cartesian product): r x s (r and s are two relations) combines information from any two tables, schema of the result is the combined schemas of the two relations

Rename: $\rho_x(E)$ renames the result of expression E to x

The Outer Join – including the null matches: extensions of the natural join operation; computes the natural join and then adds the tuples from one relation that do not match the other relation; pads the tuples with null values

Aggregate Functions - Group by/Having/..: Min, max, sum, count, average

Division

Natural Join: r ⋈ s take the cross product of two tables, select the rows you care about; combine certain SELECTIONS and a CARTESIAN PRODUCT into one operation; removes duplicates

Combinations of the above… Be able to both write and interpret queries

Entity-Relationship Diagrams

Entity-Relationship Model is used to design a database

Consists of entities and relationships

Entity: Any object that is distinguishable from any other (desk)

- o Have attributes
- o Entity Set is a set of entities of the same type
- o Value Domain of Attributes: set of permitted values for each attribute
- o Simple or composite: char, int, etc. or name with first, middle, last
- o Single values or Multiple Values: multiple phone numbers for a person
- o Derived: (Age can be found from DOB and today's date)
- o Null applies here too

Relationship: an association between 2 or more entities

- o Mathematical
- o Two important properties: participation and cardinality
- o Participation: who is involved?
- o Cardinality: how are they involved together?
- o Total Participation: every entity in the set participates in at least one relation, e.g. Every loan

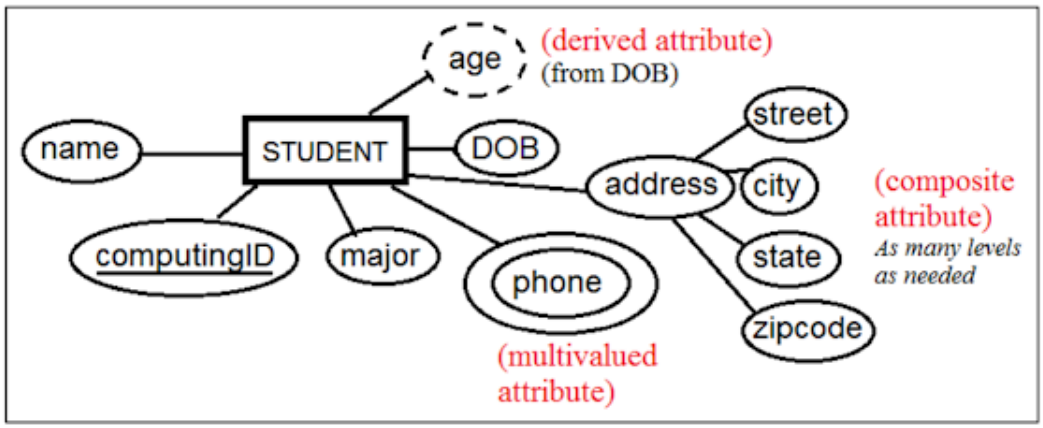
MUST have a borrower – double edge

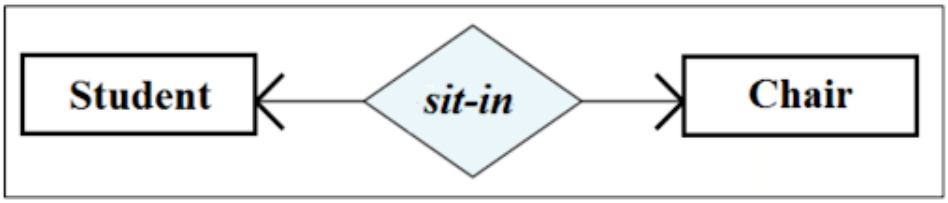
- o Partial Participation: can have entities not participating – single edge
- o Cardinality: 1 to 1, 1 to many, many to 1, many to many
- o Relationships can have their own attributes: Descriptive Attr. deposit-date
- o Recursive Relationship – Works-For with people

Basic E-R Diagram Drawing

- o Rectangle – Entity Sets
- o Diamond – Relationship Sets
- o Lines – Cardinality
- o Ellipses – Attributes
- o Double Ellipse – Multi-valued
- o Dashed Ellipse – Derived
- o Arrow – “one” relationship
- o Undirected – “many” relationship

Recall the following diagrams:





Relationship "**sit-in**" between entities "**Student**" and "**Chair**".
"Only one person will sit in one chair" (*Cardinality*: one-to-one)

What makes a good entity set?
What makes a good relationship?
Design Decisions

o Entity sets vs. attributes – person & telephone #
o Rule of thumb – if you need more info about it, make it an entity – if it is the info, make it an attribute
o Entity sets vs. Relationship sets
o Binary vs. n-ary Relationship sets
- N-ary relationship sets
    - When 3 or more things come together (hires relation in the HW2)
        - Three or more strong entity sets coming together through one diamond relation

o Strong vs. Weak entity sets
- How to set the difference
    - Does the thing I'm calling a primary key? Is it unique to be the primary key of that set?
    - Can make a weak entity set into a strong one by artificially making a primary key attribute
    - Unless that's the only thing I can identify the HW with, it's gonna be a weak entity set?

- It doesn't have a primary key, so it is dependent on the strong entity it is associated with
- There must be total participation between the weak entity set and its identifying relationship

Reduction of an E-R Schema to Tables

o Primary keys allow entity sets and relationship sets to be expressed uniformly as tables which represent the contents of the database.
o A database which conforms to an E-R diagram can be represented by a collection of tables.
o For each entity set and relationship set there is a unique table which is assigned the name of the corresponding entity set or relationship set.
o Each table has a number of columns (generally corresponding to attributes), which have unique names.
o Converting an E-R diagram to a table format is the basis for deriving a relational database design from an E-R diagram.
o Schema statements:
    o TableName (attr1, attr2, attr3, attr4) -or- TableName (attr1 int, attr2 varchar, attr3 date)
    o See "Relational Schema for Bank Enterprise.pdf"

SQL

Review the SQL cheat sheet and guide; also look at one of the provided .sql files (e.g. alldbs.sql)
RA was just for query and manipulation; SQL does: table/schema definition, query and modification,
transaction control, embedded SQL, integrity constraints, authorization
Select From == Project Where == project and select
University Database / Bank Database / Sailors Database
Division in SQL will **not** be covered on the exam (BUT, **division in RA might be!**)
Aggregate functions - having and group by
Sorting - asc and desc
Set membership - in / not in / where exists / where not exists
Insert/Update/Delete
Domain datatypes - create type
    • Good reference: http://dev.mysql.com/doc/refman/5.7/en/data-types.html
    • Nice summary: http://www.tutorialspoint.com/mysql/mysql-data-types.htm
Integrity constraints - not null, unique
Creating foreign keys with references

- Linkage of something in one table with something in another table
- The other thing is not always the pk of the other
-

Views

Advanced SQL commands ("building business logic")

(No code. Know what they do and how they're useful. Give examples. 1-2 sentences)

o Stored Procedures

1. it's implementation is the most different from one database to another
2. It's simply a function call and is a stored select or insert where you can pass parameters to it
3. -Ex. "GiveEveryoneARaise" stored procedure: find all employees at some company, apply some formula to give everyone a raise according to some pre-set logic, you can pass in parameters like 0.2 or only managers, etc

o Triggers

1. Event - driven, reacting updates, deletes, or insertions. Once this event is detected, they "do something" usually performing a particular query"
2. Triggers are really where you start building in business logic
3. Ex. Warehouse: every time a shipment goes out, database updates inventory. If inventory level drops below a certain threshold, a trigger is activated

o Assertions

1. One-table, multi-attribute, it could involve a second table
2. Usually more complex and more powerful than Checks
3. Ex. The sum in another table must be greater than the min in this table
4. Ex 2. You are only allowed to withdraw up to $200/day from an ATM

o Checks

1. Truth statement that you make about a single field in one table that is "checked" any time there is an insert or an update on that table
2. Ex. Check balance > 0: for instance, in Bank database, add a check constraint "Balance of someone's account cannot go below zero"

o Integrity Constraints

1. Making sure data in your DB is valid (data integrity)
2. Ex. Foreign Keys, NULL or not (Will you allow computing ID to be NULL?), Default values, primary keys

o Data Types

1. The simplest way to control what kind of data is in your database
2. Forces all data populating a particular attribute to look a certain way
3. A very low level, easy way, of ensuring the data is one type
4. Ex. :Integers, varchar, text, date, decimals, boolean, enum, etc.

Others

ER Diagram conversion to tables

- Many-to-many relationship: attributes are primary keys of participating entity sets and any attributes on the relationship itself, and primary key is primary keys of the entity sets
- one-to-many/many-to-one:
    - attributes are primary keys from many side and one side and primary key is primary key of both sides?
- For multivalued attributes, make the another entity name in addition to the original entity name originalentity_multivaluedattribute (primarykey, multivaluedattribute) and don't put the multivalued attribute as the attribute for the original entity
    - Double circle for multivalued attribute in ER diagram
- ISA: primary key of generalization (ex. Hospital staff) goes inside the entity sets of specialization (ex. Doctor, nurse) as primary key attribute