Feb 4: Recap of Jan 30 class

- Data Models: E-R and Relational (and some others of mostly historical interest)
- We examined the E-R model
  - Entities, Relationships, and Attributes
  - Diagram-based model
- Talked about Keys
- Some special cases for the E-R model
- Today we’ll examine two enhancements of the E-R model that allow representation of some hierarchical information, then move on to the Relational database model.
Specialization-Generalization (ISA Hierarchy)

• This is a way to represent entity complexity

• specialization: top-down refinement of entities with distinct attributes
  – Entity type BANK ACCOUNT might be subdivided into related but different types CHECKING ACCT and SAVINGS ACCT

• generalization: bottom-up abstraction of common attributes
  – Course types DATABASE, SYSTEM, and NETWORK all have common attribute (project). From them we can abstract a new course type PRACTICAL COURSE
  – other common course attributes are included (e.g., course number)
ISA Hierarchy Example: Top-down Refinement

- *Account* entity with attributes *balance* and *number*
- additional complexity: we want to represent two subtypes of account
  - *Savings Account* with attribute *Interest Rate*
  - *Checking Account* with attribute *Overdraft Limit*
ISA Hierarchy Example: Bottom-up Abstraction

- Three related entities with similar attribute *project*
- we abstract a new type of super entity *Practical Course* and link the three entities as subtypes
- other shared attributes (e.g., *course number*) are also promoted to the upper level entity
Aggregation
(Part-of Hierarchy)

• This is a way to represent relationship complexity
  – relationships among relationships are not supported by the E-R model
  – often we want to model lower-level relationships differently

• Groups of entities and relationships can be abstracted into higher level entities
Part-of Hierarchy Example

- Entities *driver, car, tires, doors, engine, seats, piston, valves*
- Relationship *drives* is insufficient to model the complexity of this system
- *Part-of* relationships allow abstraction into higher level entities (*piston* and *valves* as parts of *engine*; *engine, tires, doors, seats* aggregated into *car*)
Mapping an E-R Schema to Tables

• Motivation - translating E-R database designs into Relational designs
  – Both models are abstract, logical representations of a real-world enterprise
  – Both models employ similar design principles
  – Converting an E-R diagram to tables is the way we translate an E-R schema to a Relational schema.
  – Later on we’ll examine how to convert a Relational schema to an E-R schema
Mapping an E-R Schema to Tables (2)

- Strong Entity E with primary key PK and attributes A, B, …
  ==> \( E(PK, A, B, \ldots) \)
- Weak Entity F with (non-primary) key WK and attributes C, D, …
  depending upon E above for primary key
  ==> \( F(PK, WK, C, D, \ldots) \)
- Relationship R with attributes L, M, … and associating Entities E
  (with primary key PK), E2 (PK2), E3 (PK3), …
  ==> \( R(PK, PK2, PK3, \ldots, L, M, \ldots) \)
- Relationships between weak entities and the strong one on which they are
  dependent usually do not require representation because it is usually a
  many-one relationship with no attributes on the relationship (they are on
  the weak entity) and so the resulting table \( R(PK, WK) \) is a subset of the
  weak entity itself.
Table Details

• The whole table represents a single Entity Set or Relationship Set.
• Each entry (row) in the table corresponds to a single instance (member in that set)
• For an Entity Set each column in the table represents an attribute in the E-R diagram
• For a Relationship Set each column in the table represents either an attribute of the Relationship or one of the parts of the primary key of the Entity Sets it associates
Mapping an E-R Schema to Tables (3)

– ISA relationships: choose either to
  • Represent the super class entity, then represent each subclass with the primary key of the super class and its own attribute set. This is very similar to the way weak entities are treated.
  • Or, map the subclasses to separate relations and ignore the whole super class. This is good when the subclasses partition the whole superclasses between them (the subclasses are disjoint and the union of the subclasses covers the whole super class).

– Aggregate (part-of) relationship
  • Translation is straightforward -- just treat the aggregate as an entity and use the methods defined above.

– With last week’s lecture, this covers the material of chapter 2.
Relational Database Model

• Most popular logical data model
• Relations (also called tables) represent both Entity Sets and Relationship Sets.
  – Attributes form the columns of the table (*column* and *attribute* are synonymous)
  – Each row represents a single entity or relationship (called a *row* or *tuple*)
• Each instance of an attribute takes values from a specific set called the *domain* of the column (the domain defines the *type*)
Relational Database Model (cont)

- A *relation schema* is made up of the name and attributes of a relation with their underlying domains.
- A *database schema* is a set of all relation schemas.
- The notions of keys, primary keys, superkeys are all as previously described.
Query Languages

• a language in which a user requests information from the database
  – a higher level language than standard programming languages

• query languages may be *procedural* or *non-procedural*
  – procedural languages specify a series of operations on the database to generate the desired result
  – non-procedural languages do not specify how the information is generated
  – most commercial relational database systems offer a query language that includes procedural and non-procedural elements
Relational Algebra

- procedural query language
- set of operators that map one or more relations into another relation
- closed algebraic system
  - best feature - operations on operations
  - form relational algebraic expressions
- two types of operations: set-theoretic and database specific
Relational Algebra Operations

- **database specific:**
  - (horizontal) selection (\(？\))
  - (vertical) projection (\(?\))
  - join
  - outer join
  - semijoin
  - division

- **set operators**
  - union
  - difference
  - intersection
  - cartesian (cross) product
### Example Relations

<table>
<thead>
<tr>
<th>EMP</th>
<th>ename</th>
<th>salary</th>
<th>dept</th>
<th>DEPT</th>
<th>dept</th>
<th>floor</th>
<th>mgr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gary</td>
<td>30K</td>
<td>toy</td>
<td></td>
<td>candy</td>
<td>1</td>
<td>Irene</td>
<td></td>
</tr>
<tr>
<td>Shirley</td>
<td>35K</td>
<td>candy</td>
<td></td>
<td>toy</td>
<td>2</td>
<td>Jim</td>
<td></td>
</tr>
<tr>
<td>Christos</td>
<td>37K</td>
<td>shoe</td>
<td></td>
<td>men</td>
<td>2</td>
<td>John</td>
<td></td>
</tr>
<tr>
<td>Robin</td>
<td>22K</td>
<td>toy</td>
<td></td>
<td>shoe</td>
<td>1</td>
<td>George</td>
<td></td>
</tr>
<tr>
<td>Uma</td>
<td>30K</td>
<td>shoe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tim</td>
<td>12K</td>
<td>(null)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Database Specific Operators

• (horizontal) selection (?)
  – picks a subset of the rows

• (vertical) projection (?)
  – picks a subset of the columns

• join
  – creates a new relation (table) out of two
    • equijoin (based upon equality of attributes)
    • natural join (equijoin plus projection to eliminate duplicated columns)
Set Operators

• union
  – both relations must be union-compatible -- same degree and same domains

• set difference
  – both relations must be union-compatible as above

• intersection
  – same deal

• cartesian (cross) product
  – note similarity to join operation; join can be defined as a cross product followed by a selection criteria
More Operators

• rename (?)
  – results of operations in the relational algebra do not have names
  – it is often useful to be able to name such results for use in further expressions later on
  – conceptually similar to an assignment operator in most programming languages

• semijoin
  – very useful in practical implementation of large queries
  – semijoin of R and S is equivalent to the join of R and S projected onto the attributes of R.