Final Exam Overview

- 6 hours, multiple sittings
  - Open book, notes, MySQL database, etc. (the usual)
- Primary topics: everything in the last half of the term
  - DB schema design and Entity-Relationship Model
  - Functional/multivalued dependencies, normal forms
  - Also SQL DDL, DML, stored routines, hierarchies, etc.
- Questions will generally take this form:
  - “Design a database to model such-and-such a system.”
    - Create an E-R diagram for the database
    - Translate to relational model and DDL
    - Write some queries and/or stored routines against your schema
  - Functional/multivalued dependency problems as well
Final Exam Admin Notes

- Final exam will be available towards end of week
- **Due next Wednesday, December 7 at 5:00 pm**
- Solution sets for all assignments will be available by the end of the week
- (Ideally, HW5 and HW6 will be graded before the exam, but no promises...)

Entity-Relationship Model

- Diagramming system for specifying DB schemas
  - Can map an E-R diagram to the relational model
- Entity-sets (a.k.a. strong entity-sets)
  - “Things” that can be uniquely represented
  - Can have a set of attributes; **must** have a primary key
- Relationship-sets
  - Associations between two or more entity-sets
  - Can have descriptive attributes
  - Relationships in a relationship-set are uniquely identified by the participating entities, **not** the descriptive attributes
  - Primary key of relationship depends on mapping cardinality of the relationship-set
Entity-Relationship Model (2)

- **Weak entity-sets**
  - Don’t have a primary key; have a discriminator instead
  - Must be associated with a strong entity-set via an identifying relationship
  - Diagrams must indicate both weak entity-set and the identifying relationship(s)

- **Generalization/specialization of entity-sets**
  - Subclass entity-sets inherit attributes and relationships of superclass entity-sets

- Schema design problems will likely involve most or all of these things in one way or another
E-R Model Guidelines

- You should know:
  - How to properly diagram each of these things
  - Various constraints that can be applied, what they mean, and how to diagram them
  - How to map each E-R concept to the relational model
    - Including rules for primary keys, candidate keys, etc.
- Final exam problem will require familiarity with all of these points
- Make sure you are familiar with the various E-R design issues, so you don’t make those mistakes!
E-R Model Attributes

- Attributes can be:
  - Simple or composite
  - Single-valued or multivalued
  - Base or derived

- Attributes are listed in the entity-set’s rectangle
  - Components of composite attributes are indented
  - Multivalued attributes are enclosed with { }
  - Derived attributes have a trailing ()

- Entity-set primary key attributes are underlined
- Weak entity-set partial key has dashed underline
- Relationship-set descriptive attributes aren’t a key!
Example Entity-Set

- **customer** entity-set

- **Primary key:**
  - `cust_id`

- **Composite attributes:**
  - `name`, `address`

- **Multivalued attribute:**
  - `phone_number`

- **Derived attribute:**
  - `age`

<table>
<thead>
<tr>
<th>customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_id</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>first_name</td>
</tr>
<tr>
<td>middle_initial</td>
</tr>
<tr>
<td>last_name</td>
</tr>
<tr>
<td>address</td>
</tr>
<tr>
<td>street</td>
</tr>
<tr>
<td>city</td>
</tr>
<tr>
<td>state</td>
</tr>
<tr>
<td>zip_code</td>
</tr>
<tr>
<td>{ phone_number }</td>
</tr>
<tr>
<td>birth_date</td>
</tr>
<tr>
<td>age ()</td>
</tr>
</tbody>
</table>
Example Relationship-Set

- Relationships are identified only by participating entities
  - Different relationships can have same value for a descriptive attribute

- Example:

  A given pair of customer and loan entities can only have one relationship between them via the borrower relationship-set
E-R Model Constraints

- E-R model can represent several constraints:
  - Mapping cardinalities
  - Key constraints in entity-sets
  - Participation constraints

- Make sure you know when and how to apply these constraints

- Mapping cardinalities:
  - “How many other entities can be associated with an entity, via a particular relationship set?”
  - Choose mapping cardinality based on the rules of the enterprise being modeled
Mapping Cardinalities

- In relationship-set diagrams:
  - arrow towards entity-set represents “one”
  - line with no arrow represents “many”
  - arrow is always towards the entity-set

- Example: many-to-many mapping
  - The way that most banks work…

```plaintext
<table>
<thead>
<tr>
<th>customer</th>
<th>access_date</th>
<th>loan</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_id</td>
<td></td>
<td>loan_id</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td>amount</td>
</tr>
<tr>
<td>street_address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>city</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

borrower

11
Mapping Cardinalities (2)

- **One-to-many mapping:**

  - **customer**
    - cust_id
    - name
    - street_address
    - city

  - **borrower**

  - **access_date**

  - **loan**
    - loan_id
    - amount

- **One-to-one mapping:**

  - **customer**
    - cust_id
    - name
    - street_address
    - city

  - **borrower**

  - **access_date**

  - **loan**
    - loan_id
    - amount
Relationship-Set Primary Keys

- Relationship-set $R$, involving entity-sets $A$ and $B$
- If mapping is many-to-many, primary key is:
  \[ \text{primary_key}(A) \cup \text{primary_key}(B) \]
- If mapping is one-to-many, $\text{primary_key}(B)$ is primary key of relationship-set
- If mapping is many-to-one, $\text{primary_key}(A)$ is primary key of relationship-set
- If mapping is one-to-one, use $\text{primary_key}(A)$ or $\text{primary_key}(B)$ for primary key
  - Enforce both as candidate keys in the implementation schema!
Participation Constraints

- Given entity-set $E$, relationship-set $R$
- If every entity in $E$ participates in at least one relationship in $R$, then:
  - $E$’s participation in $R$ is **total**
- If only some entities in $E$ participate in relationships in $R$, then:
  - $E$’s participation in $R$ is **partial**
- Use total participation when enterprise requires all entities to participate in at least one relationship
Diagramming Participation

- Can indicate participation constraints in entity-relationship diagrams
  - Partial participation shown with a single line
  - Total participation shown with a double line
Weak Entity-Sets

- Weak entity-sets don’t have a primary key
  - Must be associated with an identifying entity-set
  - Association called the identifying relationship
  - If you use weak entity-sets, make sure you also include both of these things!

- Every weak entity is associated with an identifying entity
  - Weak entity’s participation in relationship-set is total

- Weak entities have a discriminator (partial key)
  - Need to distinguish between the weak entities
  - Weak entity-set’s primary key is partial key combined with identifying entity-set’s primary key
Diagramming Weak Entity-Sets

- In E-R model, can only tell that an entity-set is weak if it has a discriminator instead of a primary key
  - Discriminator attributes have a dashed underline
- Identifying relationship to owning entity-set indicated with a double diamond
  - One-to-many mapping
  - Total participation on weak entity side

<table>
<thead>
<tr>
<th>account</th>
<th>check</th>
</tr>
</thead>
<tbody>
<tr>
<td>account_number</td>
<td>check_number</td>
</tr>
<tr>
<td>balance</td>
<td>check_date</td>
</tr>
<tr>
<td></td>
<td>recipient</td>
</tr>
<tr>
<td></td>
<td>amount</td>
</tr>
<tr>
<td></td>
<td>memo</td>
</tr>
</tbody>
</table>
Weak Entity-Set Variations

- Can run into interesting variations:
  - A strong entity-set that owns several weak entity-sets
  - A weak entity-set that has multiple identifying entity-sets

- Example:

  - Other (possibly better) ways of modeling this too, e.g. make submission a strong entity-set with its own ID

- Don’t forget: weak entity-sets can also have their own non-identifying relationship-sets, etc.
Conversion to Relation Schemas

- Converting strong entity-sets is simple
  - Create a relation schema for each entity-set
  - Primary key of entity-set is primary key of relation schema

- Components of compound attributes are included directly in the schema
  - Relational model requires atomic attributes

- Multivalued attributes require a second relation
  - Includes primary key of entity-set, and “single-valued” version of attribute

- Derived attributes normally require a view
  - Must compute the attribute’s value
Global Schema Example

- **customer entity-set:**

<table>
<thead>
<tr>
<th>customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_id</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>address</td>
</tr>
<tr>
<td>street</td>
</tr>
<tr>
<td>city</td>
</tr>
<tr>
<td>state</td>
</tr>
<tr>
<td>zip_code</td>
</tr>
</tbody>
</table>

- Maps to schema:

  \[
  \text{customer}(\text{cust_id}, \text{name}, \text{street}, \text{city}, \text{state}, \text{zipcode}) \\
  \text{customer_emails}(\text{cust_id}, \text{email})
  \]

- Primary-key attributes come first in attribute lists!
Bank loans:

Maps to schema:

\[ \text{customer}(\text{cust}_\text{id}, \text{name}, \text{street}_\text{address}, \text{city}) \]

\[ \text{loan}(\text{loan}_\text{id}, \text{amount}) \]

\[ \text{borrower}(\text{loan}_\text{id}, \text{cust}_\text{id}, \text{access}_\text{date}) \]
Checking accounts:

- **account**: account_number, balance
- **check**: account_number, check_number, check_date, recipient, amount, memo

No schema for identifying relationship!

Maps to schema:

- account(account_number, balance)
- check(account_number, check_number, check_date, recipient, amount, memo)
Generalization and Specialization

- Use generalization when multiple entity-sets represent similar concepts
- Example: checking and savings accounts

- Attributes and relationships are inherited
  - Subclass entity-sets can also have own relationships
Specialization Constraints

- Disjointness constraint, a.k.a. disjoint specialization:
  - Every entity in superclass entity-set can be a member of at most one subclass entity-set.
  - One arrow split into multiple parts shows disjoint specialization.

- Overlapping specialization:
  - An entity in the superclass entity-set can be a member of zero or more subclass entity-sets.
  - Multiple separate arrows show overlapping specialization.

Diagram:
- person
  - ID
  - name
  - address
- employee
  - salary
- student
  - tot_credits
- instructor
  - rank
- secretary
  - hours_per_week
Specialization Constraints (2)

- Completeness constraint:
  - Total specialization: every entity in superclass entity-set must be a member of some subclass entity-set
  - Partial specialization is default
  - Show total specialization with “total” annotation on arrow

- Membership constraint:
  - What makes an entity a member of a subclass?
  - Attribute-defined vs. user-defined specialization
Generalization Example

- Checking and savings accounts:

- One possible mapping to relation schemas:
  - account(acct_id, acct_type, balance)
  - checking(acct_id, overdraft_limit)
  - savings(acct_id, min_balance, interest_rate)

- Be familiar with other mappings, and their tradeoffs
If all subclass entity-sets have a relationship with a particular entity-set:

- e.g. all accounts are associated with customers
- Don’t create a separate relationship for each subclass entity-set!

Creates unnecessary complexity in the database schema.
If all subclass entity-sets have a relationship with a particular entity-set:

- Create a relationship with superclass entity-set
- Subclass entity-sets inherit this relationship

Both checking and savings accounts inherit relationships with customers.
Finally, ask yourself:

- “What constraints should I enforce on depositor?”
- All accounts have to be associated with at least one customer
- A customer may have zero or more accounts
- account has total participation in depositor
- Subclass entity-sets can have their own relationships
  - e.g. associate every checking account with one specific “overdraft” savings account
  - What constraints on overdraft?

In this specific case, could also make `overdraft_limit` a descriptive attribute on `overdraft`. 
Normal Forms

- Normal forms specify “good” patterns for database schemas
- First Normal Form (1NF)
  - All attributes must have atomic domains
  - Happens automatically in E-R to relational model conversion
- Second Normal Form (2NF) of historical interest
  - Don’t need to know about it
- Higher normal forms use more formal concepts
  - Functional dependencies: BCNF, 3NF
  - Multivalued dependencies: 4NF
Normal Form Notes

- Make sure you can:
  - Identify and state functional dependencies and multivalued dependencies in a schema
  - Determine if a schema is in BCNF, 3NF, 4NF
  - Normalize a database schema

- Functional dependency requirements:
  - Apply rules of inference to functional dependencies
  - Compute the closure of an attribute-set
  - Compute $F_c$ from $F$, without any programs this time 😊
  - Identify extraneous attributes
Functional Dependencies

- Given a relation schema $R$ with attribute-sets $\alpha$, $\beta \subseteq R$
  - The functional dependency $\alpha \rightarrow \beta$ holds on $r(R)$ if
    \[ \forall t_1, t_2 \in r : t_1[\alpha] = t_2[\alpha] : t_1[\beta] = t_2[\beta] \]
  - If $\alpha$ is the same, then $\beta$ must be the same too

- Trivial functional dependencies hold on all possible relations
  - $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

- A superkey functionally determines the schema
  - $K$ is a superkey if $K \rightarrow R$
Inference Rules

- Armstrong’s axioms:
  - Reflexivity rule:
    If $\alpha$ is a set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.
  - Augmentation rule:
    If $\alpha \rightarrow \beta$ holds, and $\gamma$ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.
  - Transitivity rule:
    If $\alpha \rightarrow \beta$ holds, and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.

- Additional rules:
  - Union rule:
    If $\alpha \rightarrow \beta$ holds, and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.
  - Decomposition rule:
    If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.
  - Pseudotransitivity rule:
    If $\alpha \rightarrow \beta$ holds, and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds.
Sets of Functional Dependencies

- A set $F$ of functional dependencies
- $F^+$ is closure of $F$
  - Contains all functional dependencies in $F$
  - Contains all functional dependencies that can be logically inferred from $F$, too
  - Use Armstrong’s axioms to generate $F^+$ from $F$
- $F_c$ is canonical cover of $F$
  - $F$ logically implies $F_c$, and $F_c$ logically implies $F$
  - No functional dependency has extraneous attributes
  - All dependencies have unique left-hand side
- Review how to test if an attribute is extraneous!
Boyce-Codd Normal Form

- Eliminates all redundancy that can be discovered using functional dependencies

- Given:
  - Relation schema $R$
  - Set of functional dependencies $F$

- $R$ is in BCNF with respect to $F$ if:
  - For all functional dependencies $\alpha \rightarrow \beta$ in $F^+$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
    - $\alpha \rightarrow \beta$ is a trivial dependency
    - $\alpha$ is a superkey for $R$

- Is not dependency-preserving
  - Some dependencies in $F$ may not be preserved
A dependency-preserving normal form
Also allows more redundant information than BCNF

Given:
Relation schema $R$, set of functional dependencies $F$

$R$ is in 3NF with respect to $F$ if:

For all functional dependencies $\alpha \rightarrow \beta$ in $F^+$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is a trivial dependency
- $\alpha$ is a superkey for $R$
- Each attribute $A$ in $\beta - \alpha$ is contained in a candidate key for $R$

Can generate a 3NF schema from $F_c$
Multivalued Dependencies

- Functional dependencies cannot represent multivalued attributes
  - Can’t use functional dependencies to generate normalized schemas including multivalued attributes

- Multivalued dependencies are a generalization of functional dependencies
  - Represented as $\alpha \rightarrow \beta$

- More complex than functional dependencies!
  - Real-world usage is usually very simple

- Fourth Normal Form
  - Takes multivalued dependencies into account
Multivalued Dependencies (2)

- Multivalued dependency $\alpha \rightarrow \beta$ holds on $R$ if, in any legal relation $r(R)$:
  - For all pairs of tuples $t_1$ and $t_2$ in $r$ such that $t_1[\alpha] = t_2[\alpha]$
  - There also exists tuples $t_3$ and $t_4$ in $r$ such that:
    - $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
    - $t_1[\beta] = t_3[\beta]$ and $t_2[\beta] = t_4[\beta]$
    - $t_1[R - \beta] = t_4[R - \beta]$ and $t_2[R - \beta] = t_3[R - \beta]$

- Pictorially:

<table>
<thead>
<tr>
<th></th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$R - (\alpha \cup \beta)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>$a_1...a_i$</td>
<td>$a_{i+1}...a_j$</td>
<td>$a_{j+1}...a_n$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$a_1...a_i$</td>
<td>$b_{i+1}...b_j$</td>
<td>$b_{j+1}...b_n$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$a_1...a_i$</td>
<td>$a_{i+1}...a_j$</td>
<td>$b_{j+1}...b_n$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>$a_1...a_i$</td>
<td>$b_{i+1}...b_j$</td>
<td>$a_{j+1}...a_n$</td>
</tr>
</tbody>
</table>
Trivial Multivalued Dependencies

- $\alpha \rightarrow \beta$ is a trivial multivalued dependency on $R$ if all relations $r(R)$ satisfy the dependency.
- Specifically, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$, or if $\alpha \cup \beta = R$.

- Note that a multivalued dependency’s trivial-ness may depend on the schema!
  - $A \rightarrow B$ is trivial on $R_1(A, B)$, but it is not trivial on $R_2(A, B, C)$.
  - A major difference between functional and multivalued dependencies!
  - For functional dependencies: $\alpha \rightarrow \beta$ is trivial only if $\beta \subseteq \alpha$. 

Functional & Multivalued Dependencies

- Functional dependencies are also multivalued dependencies
  - If $\alpha \rightarrow \beta$, then $\alpha \rightarrow \beta$ too
  - **Additional caveat**: each value of $\alpha$ has at most one associated value for $\beta$

- Don’t state functional dependencies as multivalued dependencies!
  - Much easier to reason about functional dependencies!
Given a relation $R_1(\alpha, \beta)$ with $\alpha \rightarrow \beta$ and $\alpha \cap \beta = \emptyset$

- What is the key of $R_1$?
- $R_1(\alpha, \beta)$

Given a relation $R_2(\alpha, \beta)$ with $\alpha \rightarrow \beta$ and $\alpha \cap \beta = \emptyset$

- What is the key of $R_2$?
- $R_2(\alpha, \beta) \text{ – i.e. all attributes } \alpha \cup \beta \text{ are part of the key of } R_2$

This is why we don’t state functional dependencies as multivalued dependencies
Fourth Normal Form

- **Given:**
  - Relation schema $R$
  - Set of functional and multivalued dependencies $D$

- $R$ is in 4NF with respect to $D$ if:

  - For all multivalued dependencies $\alpha \rightarrow \beta$ in $D^+$, where $\alpha \in R$ and $\beta \in R$, at least one of the following holds:
    - $\alpha \rightarrow \beta$ is a trivial multivalued dependency
    - $\alpha$ is a superkey for $R$
  
  - **Note:** If $\alpha \rightarrow \beta$ then $\alpha \rightarrow \beta$

- A database design is in 4NF if all schemas in the design are in 4NF