Final review

Course Assessment

https://web.stevens.edu/assess/

Administrivia

- Final exam (25% of final grade)
  - BOTH CS442 and CPE442 students
  - 9:30am – 12noon, Saturday, Dec 17, B-118.

Covered Topics

- Database ER diagram design
- Relational Model
- Relational Algebra
- SQL
- Functional dependencies and Schema refinement
- Database security

Materials to Read

- If you have sufficient time, read all related chapters in textbook
- Otherwise, read all course slides on Moodle
- Re-study your assignments (and TA's comments)

Question Formats

- TRUE/FALSE questions
- Short explanation questions
- Other questions similar to midterm exam and assignments

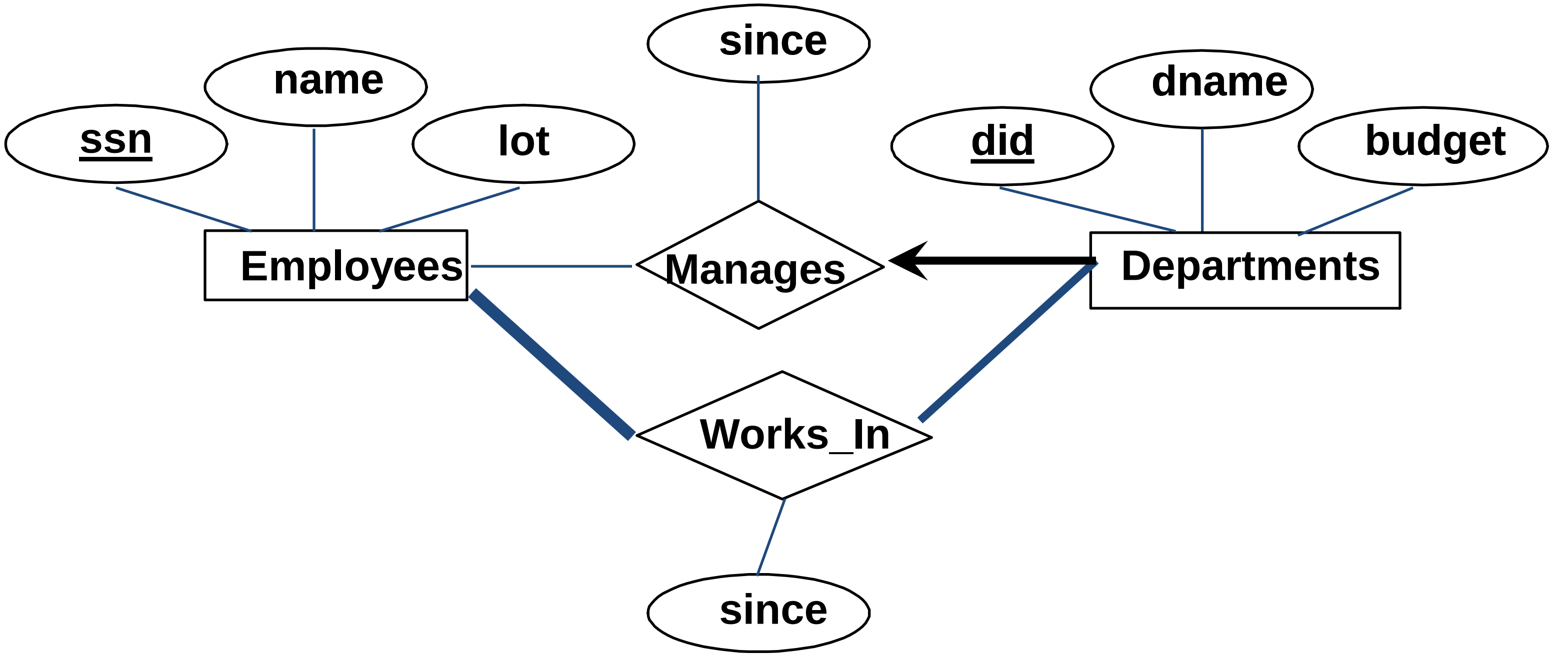
Course Overview

- Database design
- Relational model
- Relational algebra
- SQL
- Functional dependency and schema refinement

Database Design

- Entities & entity sets
  - Entities: Real-world object, distinguishable from other objects. An entity is described using a set of *attributes*.
  - Entity set: A collection of similar entities.
- Relations & relation sets
  - Relations: Association among two or more entities.
  - Relation set: Collection of similar relationships.

E-R Diagrams



- **Rectangles** represent entity sets.
- **Diamonds** represent relationship sets.
- **Lines** link attributes to entity sets and entity sets to relationship sets.
- **Ellipses** represent attributes
- **Underline** indicates primary key attributes
- **Arrow** indicates 1-to-many relationship (leaving the m side, point to the 1 side)
- **Bold line** indicates total participation relationship

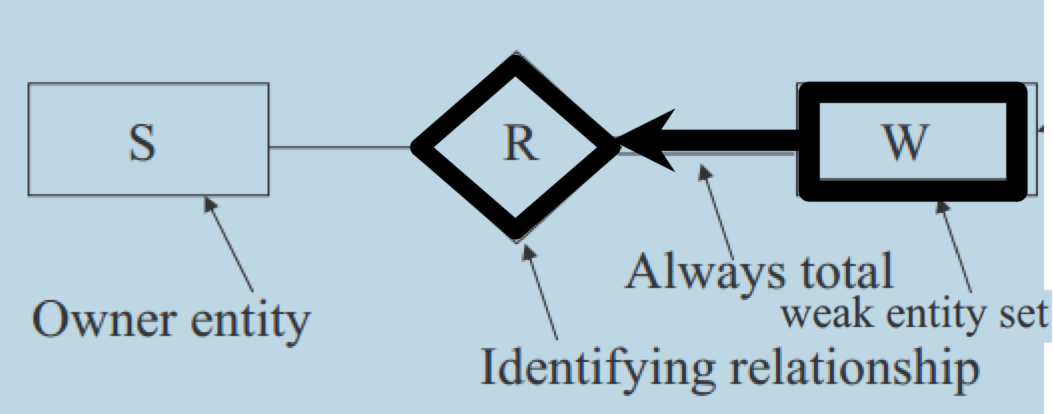
Cardinality Constraints in E-R Diagram

- Between a relationship set and an entity set
- 4 types:
  - One-to-one (1:1)
  - One-to-many (1:n)
  - Many-to-one (n:1)
  - Many-to-many (m:n)

How to represent these relationships in E-R diagram?

Participation Constraints & Weak Entities

- Participation constraint: Total/partial participation
- Weak entity: An entity set that does not have a primary key
  - A *weak entity* can be identified uniquely only by considering the primary key of another (*owner, or identifier*) entity.



ISA & Aggregation

- ISA: One entity type ISA subtype of another
  - *Inheritance* - Attributes of supertype apply to subtype.
  - *overlap/covering* for ISA hierarchies.
- Aggregation: Allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.

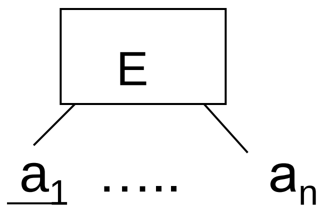
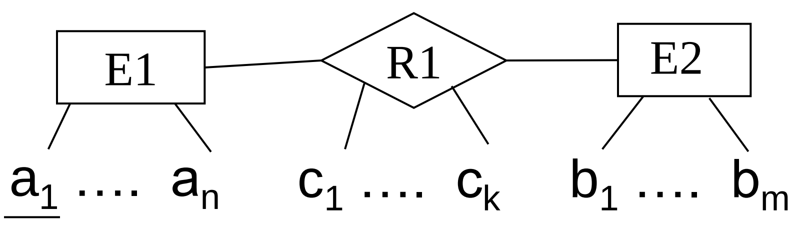
ER Design Choices

- Design choices:
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or ternary?

Relational Model

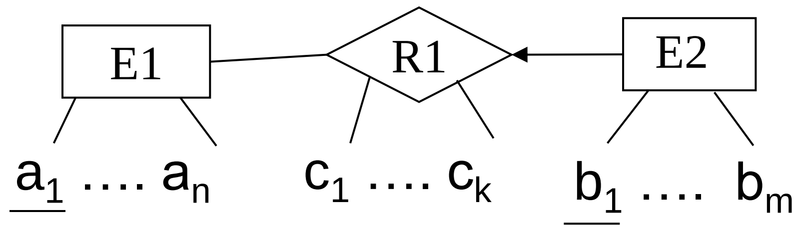
- *Relational database:* a set of *relations*.
- *Relation:* made up of 2 parts:
  - <u>*Schema*</u> : specifies the name and *attributes* of relation
  - <u>*Instance*</u> : a *table*, with rows and columns.

E/R to Relations

E/R diagram	Relational schema, e.g. account=(bname, acct_no, bal)
	E = ($\underline{a_1}$, …, a_n)
	R1= ($\underline{a_1, b_1}$, c_1, …, c_k)

More on relationships

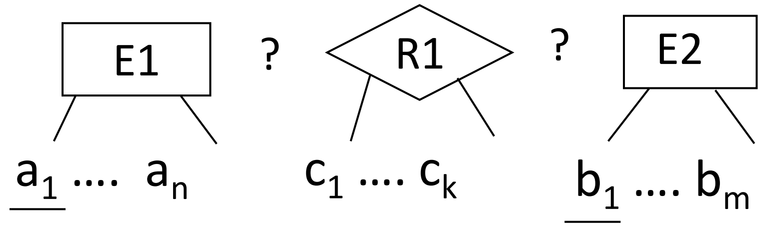
- What about:

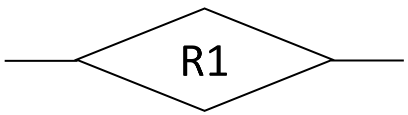
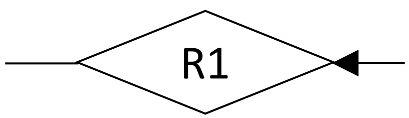
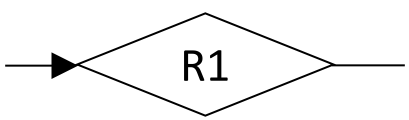
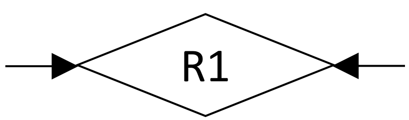


R1= ($\underline{a_1}$, $\underline{b_1}$, c_1, …, c_k)

- Could have :
  - put b_1 as the key for R1, it is also the key for E2=(b_1, …., b_n)
  - Usual strategy:
    - ignore R1
    - Add a1, c1, …., ck to E2 instead, i.e.
    - E2=($\underline{b_1}$, …., b_m, a_1, c_1, …, c_k)

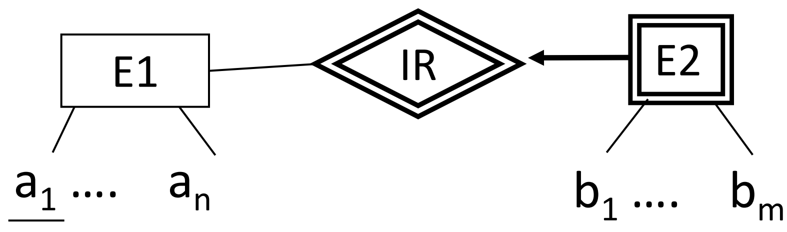
More



	E1 = ($\underline{a_1}$, …, a_n) E2 = ($\underline{b_1}$, …, b_m) R1 = ($\underline{a_1, b_1}$, c_1 …, c_k)
	E1 = ($\underline{a_1}$, …, a_n) E2 = ($\underline{b_1}$, …, b_m , a_1, c_1, …, c_k)
	E1 = ($\underline{a_1}$, …, a_n , b_1, c_1, …, c_k) E2 = ($\underline{b_1}$, …, b_m ,)
	Treat as n:1 or 1:m

E/R to Relational

- Weak entity sets



$E1 = (\underline{a_1}, \ldots, a_n)$

$E2 = (\underline{a_1, b_1}, \ldots, b_m)$

Defining a Relation Schema in SQL

- Create table
  - CREATE TABLE <name> (<field> <domain>, …)
- Insert tuples into table
  - INSERT INTO <name> (<field names>) VALUES (<field values>)
- Delete tuples from table
  - DELETE FROM <name> WHERE <condition>
- Update tuples
  - UPDATE <name>

    SET <field name1> = value1, … <field name n> = value n

    WHERE <condition>

Integrity Constraints

- Integrity constraints (ICs): conditions specified on database schema/data
- Types of ICs
  - Domain constraints: (e.g., age of students must be at least 18)
  - Keys
  - Foreign keys
  - **How to enforce keys and foreign keys in SQL**?

Relational Algebra

- *Selection* (σ) Selects a subset of *rows* from relation (horizontal).
- *Projection* (π) Retains only wanted *columns* from relation (vertical).
- *Cross-product* ($\times$) Allows us to combine two relations.
- *Set-difference* (—) Tuples in r1, but not in r2.
- *Union* ($\cup$) Tuples in r1 and/or in r2.

Compound Operators

- Joins ($\bowtie$) : compound operators involving cross product, selection, and (sometimes) projection.
  – Natural join, conditional join.
- Division (/): Useful for expressing “for all” queries like:

  Find sids of sailors who have reserved <u>all</u> boats.

SQL

- A simple SQL query has the form:

  SELECT A_1, A_2, ..., A_n
  FROM r_1, r_2, ..., r_m
  WHERE P

  - A_i represents an attribute
  - r_i represents a relation
  - P is a predicate

Nested SQL Query

- A select-from-where query that has another select-from-where query embedded within it.
  - The embedded query is called a *subquery*
  - The subquery can be nested too
  - The subquery appears within the WHERE clause
    - Can sometimes appear in the FROM clause

Aggregate Operators

- Significant extension of relational algebra.
- Operators:
  - COUNT (*)
  - COUNT ([DISTINCT] A)
  - SUM ([DISTINCT] A)
  - AVG ([DISTINCT] A)
  - MAX (A)
  - MIN (A)

Queries With GROUP BY and HAVING

```
SELECT       [DISTINCT]  target-list
FROM         relation-list
WHERE        qualification
GROUP BY     grouping-list
HAVING       group-qualification
```

- Use the HAVING clause with the GROUP BY clause to restrict which group-rows are returned in the result set

Schema Refinement

- A <u>functional dependency</u> $X \rightarrow Y$ means

  Given any two tuples in *r*, if the X values are the same, then the Y values must also be the same. (but not vice versa)
- FD Inference: new FDs can be implied by old FDS
  - Armstrong's Axioms (see FD tutorial slides for more examples)

Normal Forms

- Types: 1st, 2nd, 3rd, Boyce-Codd
- 1st $\supset$ 2nd $\supset$ 3rd $\supset$ Boyce-Codd $\supset$ ...

Boyce-Codd Normal Form (BCNF)

- BCNF: requires that the only FDs are the key constraints
- 3NF: requires that for all FD $X \rightarrow A$ in F^+
  - It is a key constraint (i.e., x is a superkey), OR
  - A is part of some candidate key (not superkey!) for R.

Decomposition into BCNF

Consider relation R with FDs F.

- First, make sure all FDs in F contain only single attribute on RHS
  - This is always doable, for example, if you have AB->CD, spit it into AB->C and AB->D);
- Next, repeat:
  - For all $X \rightarrow Y$ (in F) violates BCNF, decompose R into R - Y and XY (guaranteed to be loss-less).
- See lecture slides (including the tutorial slides) for more details

Decomposition into 3NF

Consider relation R with FDs F. Let F' be the minimal cover of F. Let R1…Rn be a lossless-join decomposition of R w.r.t. F' (can be obtained by BCNF decomposition).

(1) Identify the dependencies N in F' that is not preserved by {R1, …Rn}

(2) For each X-> A in N, create a relation schema XA and add it to {R1…Rn}

- See lecture slides (including the tutorial slides) for more details