Final review

Final exam

- 25% of final grade
- Closed book, closed cell phone
- A A4-size, 2-sided, 1-page cheat sheet.
- 1 session, 2 hours
  - Time window: 10am, Dec 11 – 8am, Dec 12.
  - The instructor will not be available during lecture time for Q&A.
- Materials to read:
  - Lecture notes on Canvas
  - In-class handouts
  - Textbook (optional)

Exam questions

Q1. True/false (cover the concepts through all chapters) (~10 minutes)

Q2. ER diagram design: Identify mistakes in a given ER diagram (you don't need to design the ER diagram by yourself) (~10 minutes)

Q3. Translate ER to relational model: Write SQL statements to create the tables for a given ER diagram (~10 minutes)

Q4. Validity of SQL queries (Yes/No): check if the given SQL queries are correct (~5 minutes)

Q5. Write relational algebra query expressions (~10 minutes)

Q6. Write SQL queries (~15 minutes)

Q7. Keys and normal forms (use FDs to find candidate keys, and check if the scheme satisfy 3NF and BCNF) (~20 minutes)

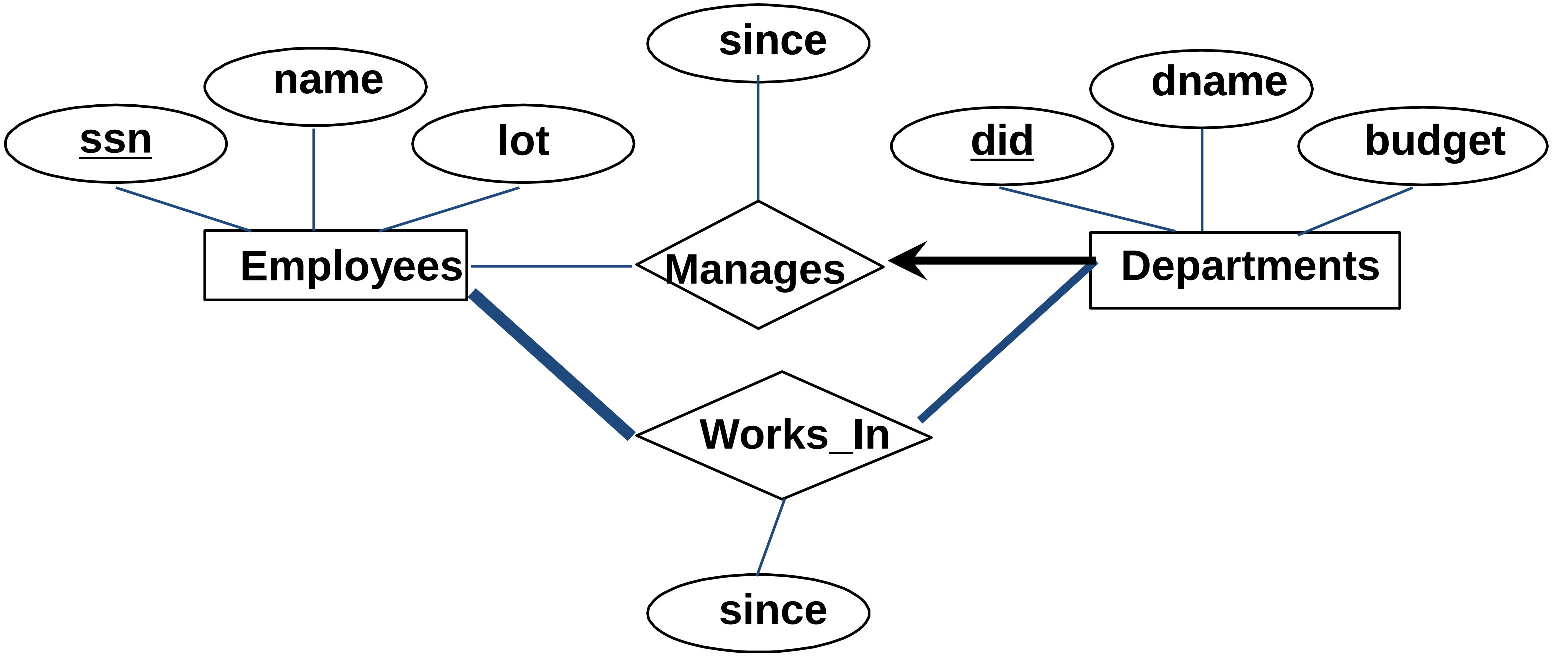
Q8. 3NF decomposition (~30 minutes)

Total: ~110 minutes

Course Overview

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

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

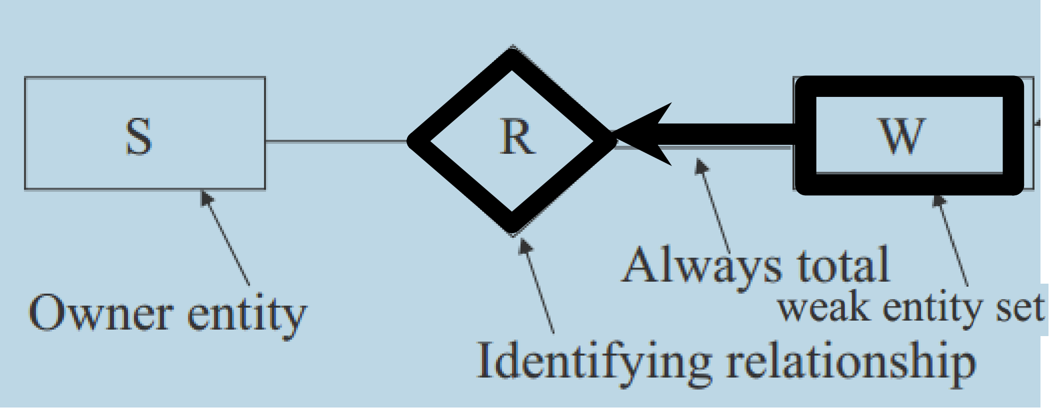
Cardinality Constraints in E-R Diagram

- 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



ISA & Aggregation

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

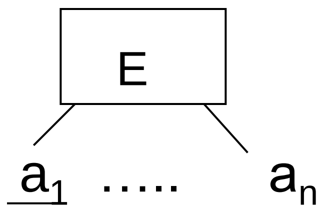
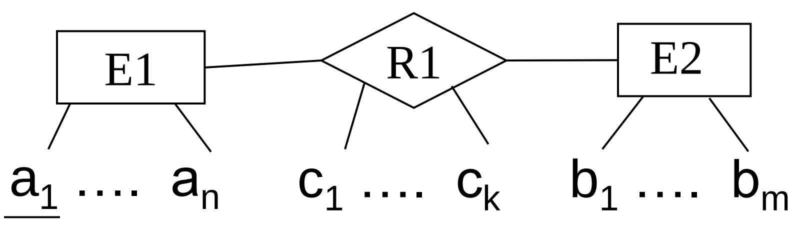
Conceptual Design Using the ER Model

- ER modeling *can* get tricky!
- <u>Design choices:</u>
  - 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

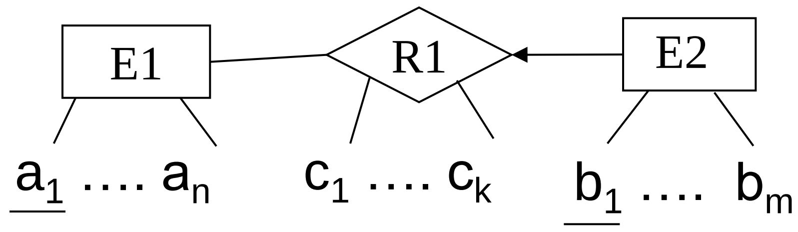
- *Relational database:* a set of *relations*.
- *Relation:* made up of 2 parts:
  - *Schema* : specifies the name and *attributes* of relation
  - *Instance* : a *table*, with rows and columns.

E/R to Relations

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

More on relationships

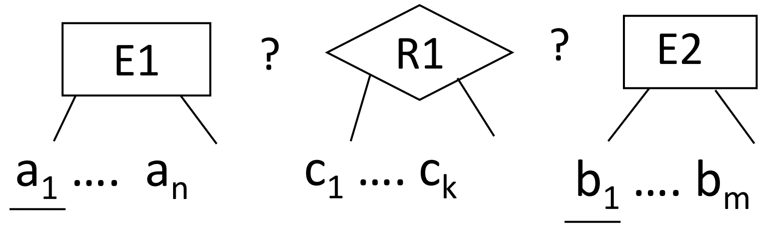
- What about:

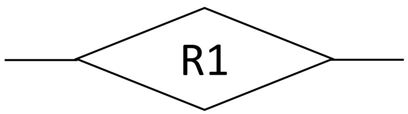
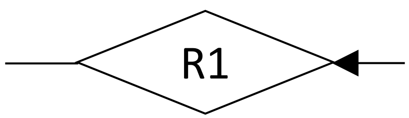
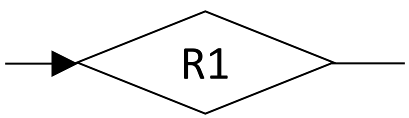
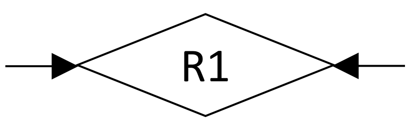


R1= ($\underline{a_1}$, $\underline{b_1}$, c_1, …, c_k)

- 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)

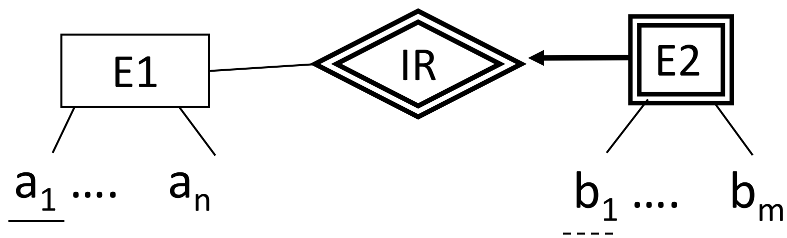
More



	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

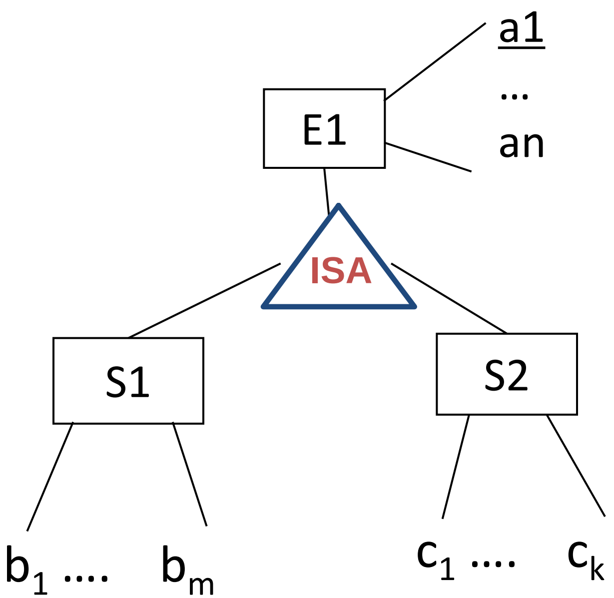


$E1 = (\underline{a_1}, \ldots, a_n)$

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

No table is needed for IR relationship

Translating ISA Hierarchies to Relations



Method 1: $E = (\underline{a}_1, \ \ldots, \ a_n)$

$S1 = (\underline{a}_1, \ b_1, \ \ldots, \ b_m)$

$S2 = (\underline{a}_1, c_1 \ldots, \ c_k)$

Method 2:

$S1 = (\underline{a}_1, \ldots, a_n, \ b_1, \ \ldots, \ b_m)$

$S2 = (\underline{a}_1, \ldots, a_n, c_1 \ldots, \ c_k)$

Defining a Relation Schema in SQL

- Create table
  - CREATE TABLE <name> (<field> <domain>, …)
  - **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* (X) Allows us to combine two relations.
- *Set-difference* (—) Tuples in r1, but not in r2.
- *Union* (U) 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 $A_1, A_2, \ldots, A_n$
  FROM $r_1, r_2, \ldots, r_m$
  WHERE P

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

- This query is equivalent to the relational algebra expression: $\prod_{A_1, A_2, \ldots, A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$

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

- FD X->Y
- FD Inference: new FDs can be implied by old FDS
  - 3 basic Armstrong's Axioms (AA rules): reflextivity, augmentation, transitivity
  - 2 advanced rules: union and decomposition
- Attribute closure

Normal Forms

- Types: 1st, 2nd, 3rd, BCNF (3.5 NF)

How to check which normal forms a relational schema satisfies?

BCNF Decomposition

Consider relation R with FDs F.

- First, make sure all FDs in F contain only single attribute on RHS
- Next, repeat:
  - For all X -> Y violates BCNF, decompose R into two tables: R - Y and XY (guaranteed to be loss-less).
- See lecture slides (including the tutorial slides) for more details

3NF Decomposition

- Consider relation R with FDs F.
- Step 1: Find the minimal cover F’ of F.
- Step 2: Obtain a BCNF decomposition {R1, …Rn} of R w.r.t. F’ (i.e., {R1, …Rn} is a lossless decomposition)
- Step 3: Identify the dependencies N in F’ that is not preserved by BCNF decomposition {R1, …Rn}
- Step 4: 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