QUIZ 1
REVIEW SESSION
DATABASE MANAGEMENT SYSTEMS
A database schema is the skeleton structure that represents the logical view of the entire database.

Logical design of a relation:

- `<relation_name>(<primary_key>, <attribute1>, <attribute2>, ...)`

Example:

- `employee(emp_id, first_name, last_name, age, date_of_hire)`
- `company(company_id, company_name, hq_location, num_employees)"
RELATIONAL ALGEBRA

- It’s basically procedural query language that takes instances of relations as input and yields relations as output
- Uses operators (unary / binary) to perform queries. Fundamental operations are as follows
  - Select (σ) – selects tuples that satisfy the given predicate
  - Project (π) – projects columns that satisfy the given predicate
  - Union (∪) – does a binary union between given relations: \( a \cup b = \{ t \mid t \in a \text{ or } t \in b \} \)
  - Set difference (−) – outputs tuples present in first relation but not the second
  - Cartesian product (⨉) – combines tuples of two relations into one relation
  - Rename (ρ) – renames output relation
RELATIONAL CALCULUS

• Tuple Relational Calculus:
  • Notation - \{T | p(T)\}
  • Example: \{ t.name | student(t) ^ t.course = ‘cs127’ \} – Returns tuples (with ‘name’) from Student who take/have taken cs127
  • TRC can be quantified using existential (\exists) or universal quantifiers (\forall)
    e.g. \{ r | \exists t \in student(t.course = ‘cs127’ ^ r.name = t.name) \} – yields same output as TRC expression above
ER MODEL

- Defines the conceptual view of a database. Think of it as a framework for organizing and interpreting data.

- Employs 3 basic concepts:
  - entity sets — set of entities (objects) of the same type that share the same properties (attributes)
  - relationship sets — set of associations (relationships) among entities
  - attributes — basically the properties of a given entity

- ER diagrams should be **simple** and **clear**
ER MODEL

• KEYS
  • Attribute(s) that uniquely identify an entity in an entity set
    • Super key – set of attributes (one or more) that collectively identify an entity
    • Candidate key – minimal super key. NOTE – entity set can have more than one candidate key
    • Primary key – candidate key chosen to uniquely identify the entity set

• RELATIONSHIPS
  • Associations among entities e.g. region sells items, employee works at company, e.t.c.
  • Number of participating relationships defines the degree of the relationship (binary (2), ternary (3), n-ary)
ER MODEL

- **Cardinality** - specifies how many instances of an entity relate to one instance of another entity.
  - **one-to-one** - one entity from A can be associated with at most one entity of B and vice versa
  - **one-to-many** - one entity from A can be associated with more than one entities of B. However, an entity from B, can be associated with at most one entity of A
  - **many-to-one** - more than one entities from A can be associated with at most one entity of B. However an entity from B can be associated with more than one entity from A
  - **many-to-many** - one entity from A can be associated with more than one entity from B and vice versa
ER DIAGRAM EXAMPLE
SQL

• Integrity Constraints
  • Primary key: **PRIMARY KEY(<attribute_name>, <attribute_name>, ...)**
    • Primary key of a table; must be non-null and unique
  • Foreign Key: **FOREIGN KEY(<attribute_name>, <attribute_name>,...) REFERENCES <table_name>**
    • References the primary key of another relation
  • non-null : <attr> NOT NULL
  • Unique: <attr> UNIQUE
    • All values must be unique (**can** include nulls)
    • Also can do: UNIQUE(<attr1>, <attr2> ...)**
SQL

• Different types of **SQL JOINs** (INNER JOIN, LEFT (OUTER) JOIN, RIGHT (OUTER) JOIN, FULL (OUTER) JOIN)

• **GROUP BY** statement - used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns

• **HAVING** clause - **WHERE** keyword cannot be used with aggregate functions

• **AND, OR** and **NOT** Operators

• **ORDER BY** Keyword
Find all the employees who work in either philosophy or history department

```sql
SELECT e.*
FROM employee e
INNER JOIN (SELECT w.e_id
              FROM works w
              INNER JOIN department d
              ON w.d_id = d.d_id
              WHERE d.dname IN ('philosophy', 'history')
              ) b
ON e.eid = b.eid
```
FUNCTIONAL DEPENDENCIES

• A set of constraints between two attributes in a relation
• Represented by an arrow sign (→): X→Y, means X functionally determines Y
• If a functional dependency X → Y holds, where Y is a subset of X, then it is called trivial. Trivial functional dependencies always hold.
• Closure ($F^+$) is the set of functional dependencies logically implied by F
• The canonical cover is a minimal set of dependencies $C$ which imply every FD defined in the closure of $F$
ARMSTRONG'S AXIOMS

- **Fundamental Rules** (W, X, Y, Z: sets of attributes)
  - **Reflexivity** - If $Y \subseteq X$ then $X \rightarrow Y$
  - **Augmentation** - If $X \rightarrow Y$ then $WX \rightarrow WY$
  - **Transitivity** - If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

- **Additional Rules**
  - **Union** - If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrowYZ$
  - **Decomposition** - If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$
  - **Pseudotransitivity** - If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$
Closure Algorithm 1

• Closure = S
• Loop
  • For each F in S, apply reflexivity and augmentation rules
  • Add the new FDs to the Closure
  • For each pair of FDs in S, apply the transitivity rule
  • Add the new Fd to Closure
• Until closure doesn't change any further
Closure Algorithm 2

algorithm \((F)\)
/* \(F\) is a set of FDs */
1. \(F^+ = \emptyset\)
2. for each possible attribute set \(X\)
3. compute the closure \(X^+\) of \(X\) on \(F\)
4. for each attribute \(A \in X^+\)
5. add to \(F^+\) the FD: \(X \rightarrow A\)
5. return \(F^+\)
Example. Assume that there are 4 attributes $A, B, C, D$, and that $F = \{A \rightarrow B, B \rightarrow C\}$. To compute $F^+$, we first get:

- $A^+ = AB^+ = AC^+ = ABC^+ = \{A, B, C\}$
- $B^+ = BC^+ = \{B, C\}$
- $C^+ = \{C\}$
- $D^+ = \{D\}$
- $AD^+ = \{A, D\}$
- $BC^+ = \{B, C\}$
- $BD^+ = BCD^+ = \{B, C, D\}$
- $ABD^+ = ABCD^+ = \{A, B, C, D\}$
- $ACD^+ = \{A, C, D\}$

It is easy to generate the FDs in $F^+$ from the closures of the above attribute sets.
Closure Example

\[ R = (A, B, C, D) \]

\[ F = \{A \rightarrow BC; C \rightarrow D\} \]
\{A\}^+ = \{A, B, C, D\} \quad \text{Minimum candidate key}

\{B\}^+ = \{B\}

\{C\}^+ = \{C, D\}

\{D\}^+ = \{D\}

\{A, B\}^+ = \{A, B, C, D\} \quad \text{Superkey}

\{A, C\}^+ = \{A, B, C, D\} \quad \text{Superkey}

\{B, C\}^+ = \{B, C, D\}

\{A, D\}^+ = \{A, B, C, D\} \quad \text{Superkey}

\{B, D\}^+ = \{B, D\}

\{C, D\}^+ = \{C, D\}

\{A, B, C\}^+ = \{A, B, C, D\} \quad \text{Superkey}

\{A, B, D\}^+ = \{A, B, C, D\} \quad \text{Superkey}

\{A, C, D\}^+ = \{A, B, C, D\} \quad \text{Superkey}

\{B, C, D\}^+ = \{B, C, D\}

\{A, B, C, D\}^+ = \{A, B, C, D\} \quad \text{Superkey}

To practice: http://raymondcho.net/RelationalDatabaseTools/RelationalDatabaseTools
Canonical Cover

A minimal set of dependencies $C$ which imply every FD defined in the closure of $F$

ALGORITHM canonical-cover($X$: FD Set)
BEGIN
  REPEAT UNTIL STABLE
  1. Where possible, apply UNION rule (A’s Axioms)
  2. Remove all extraneous attributes:
     a. Test if B extraneous in $A \rightarrow BC$
        $(B$ extraneous if $(A \rightarrow B) \in (F - \{A \rightarrow BC\} \cup \{A \rightarrow C\})^+) = F^+$
     b. Test if B extraneous in $AB \rightarrow C$
        $(B$ extraneous if $(A \rightarrow C) \in F^+)$
  END