Quiz 1 Review

What we’re covering:

- Relational Algebra
- SQL
- E-R Diagrams
- Keys and Integrity Constraints
Relational Algebra

- Be familiar with basic Relational Algebra operators, such as:
  - select \( \sigma \)
  - project \( p \)
  - union \( U \)
  - set difference \( – \)
  - cartesian product \( X \)
  - rename \( \rho \)

**Closure Property**

![Diagram of Closure Property](image)
Union ($\bigcup$)

Notation: $\text{Relation}_1 \bigcup \text{Relation}_2$

$R \bigcup S$ valid only if:

1. $R, S$ have same number of columns (arity)
2. $R, S$ corresponding columns have same name and domain (compatibility)

Example:

$\left(\pi_{\text{cname (depositor)}}\right) \bigcup \left(\pi_{\text{cname (borrower)}}\right) =$

Schema:

<table>
<thead>
<tr>
<th>Depositor</th>
<th>Borrower</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{cname}</td>
<td>\text{acct_no}</td>
</tr>
<tr>
<td>\text{cname}</td>
<td>\text{Ino}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson</td>
</tr>
<tr>
<td>Smith</td>
</tr>
<tr>
<td>Hayes</td>
</tr>
<tr>
<td>Turner</td>
</tr>
<tr>
<td>Jones</td>
</tr>
<tr>
<td>Lindsay</td>
</tr>
<tr>
<td>Jackson</td>
</tr>
<tr>
<td>Curry</td>
</tr>
<tr>
<td>Williams</td>
</tr>
<tr>
<td>Adams</td>
</tr>
</tbody>
</table>
Set Difference ( - )

\[
\begin{array}{c|c}
R & \text{R DIFFERENCE S} \\
\hline
A & 1 \\
B & 2 \\
D & 3 \\
F & 4 \\
E & 5 \\
\end{array}
\quad
\begin{array}{c|c}
B & 2 \\
F & 4 \\
E & 5 \\
\end{array}
\]

\[
\begin{array}{c|c}
S & \text{S DIFFERENCE R} \\
\hline
A & 1 \\
C & 2 \\
D & 3 \\
E & 4 \\
\end{array}
\quad
\begin{array}{c|c}
C & 2 \\
E & 4 \\
\end{array}
\]
Natural Join

Notation:  \( \text{Relation}_1 \bowtie \text{Relation}_2 \)

Idea: combines \( \rho, \times, \sigma \)

\[ r \]
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \alpha )</td>
<td>+</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>( \alpha )</td>
<td>-</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>( \alpha )</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>( \beta )</td>
<td>+</td>
<td>10</td>
</tr>
</tbody>
</table>

\[ s \]
<table>
<thead>
<tr>
<th>E</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{&quot;a&quot;} )</td>
<td>( \alpha )</td>
<td>10</td>
</tr>
<tr>
<td>( \text{&quot;a&quot;} )</td>
<td>( \alpha )</td>
<td>20</td>
</tr>
<tr>
<td>( \text{&quot;b&quot;} )</td>
<td>( \beta )</td>
<td>10</td>
</tr>
<tr>
<td>( \text{&quot;c&quot;} )</td>
<td>( \beta )</td>
<td>10</td>
</tr>
</tbody>
</table>

\[ = \]

\[ A | B | C | D | E \]
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \alpha )</td>
<td>+</td>
<td>10</td>
<td>( \text{&quot;a&quot;} )</td>
</tr>
<tr>
<td>2</td>
<td>( \alpha )</td>
<td>-</td>
<td>10</td>
<td>( \text{&quot;a&quot;} )</td>
</tr>
<tr>
<td>2</td>
<td>( \alpha )</td>
<td>-</td>
<td>20</td>
<td>( \text{&quot;a&quot;} )</td>
</tr>
<tr>
<td>3</td>
<td>( B )</td>
<td>+</td>
<td>10</td>
<td>( \text{&quot;b&quot;} )</td>
</tr>
<tr>
<td>3</td>
<td>( \beta )</td>
<td>+</td>
<td>10</td>
<td>( \text{&quot;c&quot;} )</td>
</tr>
</tbody>
</table>

\[ \text{depositor} \bowtie \bowtie \text{borrower} \]

\[ \equiv \]

\[ \pi_{\text{cname,acct_no,lno}} (\sigma_{\text{cname}=\text{cname}_2} (\text{depositor} \times \rho_{t(\text{cname}_2,lno)} (\text{borrower}))) \]
Division

Notation: $\text{Relation}_1 \div \text{Relation}_2$

Idea: expresses “for all” queries

Query: Find values for $A$ in $r$ which have corresponding $B$ values for all $B$ values in $s$
SQL

Be familiar with basic Query construction such as:

```sql
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 80000
```
SQL: Cartesian Product, Joins

Find the Cartesian product $instructor \times teaches$

```
select *
from instructor, teaches
```

Joins
```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID;
```
SQL: Exists, In, Except

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \iff r \neq \emptyset$
- **not exists** $r \iff r = \emptyset$

```
(select course_id from section where sem = 'Fall' and year = 2009)
except
(select course_id from section where sem = 'Spring' and year = 2010)
```

You can think of except as the set difference operation from Relational Algebra!
Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
    course_id not in (select course_id
                      from section
                      where semester = 'Spring' and year= 2010);
```

- note that you can use these clauses to form subqueries within your larger query
**Conceptual Design**

*The E/R Data Model*

- What is a Data Model?
  - Framework for organizing and interpreting data

**Example: E/R Data Model**

![E/R Data Model Diagram]

- **Entity₁**
  - Attribute₁₁
  - Attribute₁₂

- **Entity₂**
  - Attribute₂₁
  - Attribute₂₂
  - Attribute₂₃
E/R Data Model

Basics

- **Entities**
  - noun phrases (e.g., Bob Smith, Thayer St. Branch)
  - contained in entity sets (e.g., Employee, Branch)
  - have attributes (e.g., Employee = (essn, ename, ...))

- **Relationships**
  - verb phrases (e.g., works_at, works_for)
  - relate 2 (binary) or more (n-ary) entities
  - relationship sets characterize relationships amongst entity sets
    - e.g., (Bob Smith, Thayer St Branch) ∈ Works_At
E/R Data Model

**Keys**

- Key = set of attributes identifying individual entities or relationships

```
Employee
  essn  ename  eaddress  ephone
```

**A. Superkey:**
- any attribute set that distinguishes identities
- e.g., \{essn\}, \{essn, ename, eaddress\}

**B. Candidate Key:**
- “minimal superkey” (can’t remove attributes and preserve “keyness”)  
- e.g., \{essn\}, \{ename, eaddress\}

**C. Primary Key:**
- candidate key chosen as the key by a DBA  
- e.g., \{essn\} (denoted by underline)
# E/R Diagrams and Relations

<table>
<thead>
<tr>
<th>Relationship Cardinality</th>
<th>Relational Schema</th>
</tr>
</thead>
</table>
| n:m                      | E₁ = (a₁, ..., aₙ)  
                           | E₂ = (b₁, ..., bₘ)  
                           | R  = (a₁, b₁, c₁, ..., cₙ) |
| n:1                      | E₁ = (a₁, ..., aₙ, b₁, c₁, ..., cₙ)  
                           | E₂ = (b₁, ..., bₘ) |
| 1:n                      | E₁ = (a₁, ..., aₙ)  
                           | E₂ = (b₁, ..., bₘ, a₁, c₁, ..., cₙ) |
| 1:1                      | Treat as n:1 or 1:n |
Referential Integrity Constraints

Idea:
Prevent "dangling tuples" (e.g.: A loan with bname, Waltham when no Waltham tuple in branch)

Illustrated:

Referential Integrity:
Ensure that: Foreign Key → Primary Key value
Note: Need not ensure (i.e.: Not all branches have to have loans)

CSCI 127: Introduction to Database Systems
GOOD LUCK!

YOU GOT THIS