Outline

- Overview of modeling
- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- Entity-Relationship Model

Quiz 1

Wrong:
- It helps to improve the data security of the database.
- Physical data independence protects the physical structure of data from changes.

Right:
- Physical data independence allows the physical schema to be changed without rewriting the application program(s).
Relational Query Languages

- Example schema: \( R(A, B) \)
- Practical languages
  - **SQL**
    - select A from R where B = 5;
  - **Datalog** (sort of practical)
    - \( q(A) : - R(A, 5) \)
- Formal languages
  - **Relational algebra**
    - \( \pi_A ( \sigma_{B=5} (R) ) \)
  - **Tuple relational calculus**
    - \( \{ t : \{A\} | \exists s : \{A, B\} ( R(A, B) \land s.B = 5) \} \)
  - **Domain relational calculus**
    - Similar to tuple relational calculus

The Relational Algebra is *procedural*

- “Procedural” languages provide a set of operations
  - Each operation takes one or two relations as input, and produces a single relation as output
  - Examples: Relational Algebra

- The “declarative” languages specify the output, but don’t specify the operations
  - SQL, Relational calculus
  - Datalog (used as an intermediate layer in quite a few systems today)
Select Operation

Choose a subset of the tuples that satisfies some predicate
Denoted by $\sigma$ in relational algebra

Relation $r$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

$\sigma_{A=B \land D > 5} (r)$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

Project

Choose a subset of the columns (for all rows)
Denoted by $\Pi$ in relational algebra

Relation $r$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

$\Pi_{A,D} (r)$

<table>
<thead>
<tr>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>10</td>
</tr>
</tbody>
</table>

Relational algebra following “set” semantics – so no duplicates
SQL allows for duplicates – we will cover the formal semantics later
**Set Union, Difference**

**Relation r, s**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>α</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>α</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>α</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ r \cup s: \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
</tr>
<tr>
<td>α</td>
<td>2</td>
</tr>
<tr>
<td>β</td>
<td>1</td>
</tr>
<tr>
<td>β</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ r - s: \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
</tr>
<tr>
<td>β</td>
<td>1</td>
</tr>
</tbody>
</table>

Must be compatible schemas

What about intersection?

Can be derived

\[ r \cap s = r - (r - s); \]

---

**Rename Operation**

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.

Example:

\[ \rho_x (E) \]

returns the expression \( E \) under the name \( X \)

If a relational-algebra expression \( E \) has arity \( n \), then

\[ \rho_{x \{A_1, A_2, ..., A_n\} (E)} \]

returns the result of expression \( E \) under the name \( X \), and with the attributes renamed to \( A_1, A_2, ..., A_n \).
## Cartesian Product

Combine tuples from two relations

If one relation contains N tuples and the other contains M tuples, the result would contain N*M tuples

The result is rarely useful – almost always you want pairs of tuples that satisfy some condition

<table>
<thead>
<tr>
<th>Relation r, s</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>α</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>2</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>γ</td>
<td>10</td>
<td>b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r × s:</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>α</td>
<td>1</td>
<td>α</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>α</td>
<td>1</td>
<td>β</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>α</td>
<td>1</td>
<td>β</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>α</td>
<td>1</td>
<td>γ</td>
<td>10</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>2</td>
<td>α</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>2</td>
<td>β</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>2</td>
<td>β</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>2</td>
<td>γ</td>
<td>10</td>
<td>b</td>
</tr>
</tbody>
</table>

## Theta Join

- Combine tuples from two relations if the pair of tuples satisfies some constraint
- Equivalent to Cartesian Product followed by a Select

<table>
<thead>
<tr>
<th>Relation r, s</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>α</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>2</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>γ</td>
<td>10</td>
<td>b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r ⊘ A=C s:</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>α</td>
<td>1</td>
<td>α</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>α</td>
<td>1</td>
<td>β</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>α</td>
<td>1</td>
<td>γ</td>
<td>10</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>2</td>
<td>α</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>2</td>
<td>β</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>2</td>
<td>γ</td>
<td>10</td>
<td>b</td>
</tr>
</tbody>
</table>
Joins built on Cartesian product

- **Natural join (⋈)**
  - A Cartesian product with equality condition on common attributes
  - Example:
    - if \( r \) has schema \( R(A, B, C, D) \), and if \( s \) has schema \( S(E, B, D) \)
    - Common attributes: \( B \) and \( D \)
    - Then:
      \[
      r \bowtie s = \Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \land r.D = s.D}(r \times s))
      \]
  - **SQL Equivalent:**
    - select \( r.A, r.B, r.C, r.D, s.E \) from \( r, s \) where \( r.B = s.B \) and \( r.D = s.D \), OR
    - select * from \( r \) natural join \( s \)

Other Joins

- **Equi-join**
  - A join that only has equality conditions

- **Theta-join (⋈θ)**
  - \( r \bowtie_\theta s = \sigma_\theta(r \times s) \) (combines cartesian and select in single statement)

- Left outer join (⟕)
  - Say \( r(A, B), s(B, C) \)
  - We need to somehow find the tuples in \( r \) that have no match in \( s \)
  - What is this?
    - \( (r - \pi_{r.A,r.B}(r \bowtie s)) \)
  - \( r \bowtie s = (r \bowtie s) \cup \rho_{temp}(A, B, C) \{ (r - \pi_{r.A,r.B}(r \bowtie s)) \times \{\text{NULL}\} \} \)
**Joins Summary**

- **Tables:** $r(A, B), s(B, C)$

<table>
<thead>
<tr>
<th>name</th>
<th>Symbol</th>
<th>SQL Equivalent</th>
<th>RA expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>cross product</td>
<td>$\times$</td>
<td>select * from r, s;</td>
<td>$r \times s$</td>
</tr>
<tr>
<td>natural join</td>
<td>$\bowtie$</td>
<td>natural join</td>
<td>$\pi_{r.A, r.B, s.C}(r \times s)$</td>
</tr>
<tr>
<td>equi–join</td>
<td>$\bowtie_\theta$</td>
<td>$(\text{theta must be equality})$</td>
<td></td>
</tr>
<tr>
<td>theta join</td>
<td>$\bowtie_\theta$</td>
<td>from .. where $\theta$;</td>
<td>$\sigma_\theta(r \times s)$</td>
</tr>
<tr>
<td>left outer join</td>
<td>$r \bowtie s$</td>
<td>left outer join (with “on”)</td>
<td>(see previous slide)</td>
</tr>
<tr>
<td>full outer join</td>
<td>$r \bowtie s$</td>
<td>full outer join (with “on”)</td>
<td>--</td>
</tr>
<tr>
<td>(left) semijoin</td>
<td>$r \bowtie s$</td>
<td>none</td>
<td>$\pi_{r.A, r.B}(r \bowtie s)$</td>
</tr>
<tr>
<td>(left) antijoin</td>
<td>$r \bowtie s$</td>
<td>none</td>
<td>$r - \pi_{r.A, r.B}(r \bowtie s)$</td>
</tr>
</tbody>
</table>

**Additional Operators: Division**

- Assume $r(R), s(S)$, for queries where $S \subseteq R$:
  - $r \div s$

- Think of it as “opposite of Cartesian product”:
  - $r \div s = t \iff t \times s \subseteq r$
Relational Algebra Examples

Find all loans of over $1200:

\[ \sigma_{\text{amount} > 1200} (\text{loan}) \]

Find the loan number for each loan of an amount greater than $1200:

\[ \Pi_{\text{loan-number}} (\sigma_{\text{amount} > 1200} (\text{loan})) \]

Find names of all customers who have a loan, account, or both, from the bank:

\[ \Pi_{\text{customer-name}} (\text{borrower}) \cup \Pi_{\text{customer-name}} (\text{depositor}) \]

Find names of customers who have a loan and an account at bank:

\[ \Pi_{\text{customer-name}} (\text{borrower}) \cap \Pi_{\text{customer-name}} (\text{depositor}) \]

Find names of customers who have a loan and an account at the Perryridge branch:

\[ \Pi_{\text{customer-name}} (\sigma_{\text{branch-name} = \text{"Perryridge"}} (\sigma_{\text{borrower.loan-number} = \text{loan.loan-number}} (\text{borrower} \times \text{loan}))) \]
Relational Algebra Examples

Find *largest* account *balance*(balance), assume \{(1), (2), (3)\}

*Rename the account relation to d*

\[
\Pi_{\text{balance}}(\text{account}) - \Pi_{\text{account.balance}}(\sigma_{\text{account.balance} < d.\text{balance}}(\text{account} \bowtie_d (\text{account})))
\]

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

\[
\Pi_{F_1, F_2, \ldots, F_n}(E)
\]

- \(E\) is any relational-algebra expression
- Each of \(F_1, F_2, \ldots, F_n\) are are arithmetic expressions involving constants and attributes in the schema of \(E\).
- Given relation *instructor*(ID, name, dept_name, salary) where salary is annual salary, get the same information but with monthly salary

\[
\Pi_{ID, name, dept\_name, salary/12}(\text{instructor})
\]
Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.
  - *avg*: average value
  - *min*: minimum value
  - *max*: maximum value
  - *sum*: sum of values
  - *count*: number of values

- **Aggregate operation** in relational algebra
  
  \[ G_{i_1} G_{i_2} ... G_{i_n} \ G \ F_1(A_{i_1}), F_2(A_{i_2}),..., F_n(A_{i_n}) (E) \]

  where:
  - \( E \) is any relational-algebra expression
  - \( G_1, G_2, ..., G_n \) is a list of attributes on which to group (can be empty)
  - Each \( F_i \) is an aggregate function
  - Each \( A_i \) is an attribute name

- **Note:** Some books/articles use \( \gamma \) instead of \( G \) (Calligraphic G)

Aggregate Operation – Example

- **Relation** \( r \):

  \[
  \begin{array}{ccc}
  A & B & C \\
  \alpha & \alpha & 7 \\
  \alpha & \beta & 7 \\
  \beta & \beta & 3 \\
  \beta & \beta & 10 \\
  \end{array}
  \]

  \[
  A \ G \ \text{sum}(c) \ (r)
  \]

  \[
  \begin{array}{c}
  A \ \text{sum}(c) \\
  \alpha & 14 \\
  \beta & 13 \\
  \end{array}
  \]
Aggregate Operation – Example

- Find the average salary in each department

\[ dept\_name \ G \ \text{avg(salary)} \ (instructor) \]

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>Finance</td>
<td>80000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>History</td>
<td>62000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>87000</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
</tbody>
</table>

Result of aggregation can be re-named

- For convenience, we permit renaming as part of aggregate operation

\[ dept\_name \ G \ \text{avg(salary)} \ \text{as avg\_sal} \ (instructor) \]
Modification of the Database

• The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating

• All these operations can be expressed using the assignment operator

\[
\begin{align*}
t_{1} & \leftarrow R \times S \\
t_{2} & \leftarrow \sigma_{r.A_{1} = s.A_{1}} \land r.A_{2} = s.A_{2} \land \ldots \land r.A_{n} = s.A_{n} \left( t_{1} \right)
\end{align*}
\]

\[
\text{result} = \Pi_{R \cup S} \left( t_{2} \right)
\]

The result of R x S potentially has duplicated attributes. For example, \( r(A, B) \times s(B, C) \) results in tuples w/ attributes \{A, B, C\}: \( \Pi_{R \cup S} \) gets rid of the extra B. Duplicated tuples are an entirely different thing, and are not present in the relational algebra.

Multiset Relational Algebra

• Pure relational algebra removes all duplicates
  - e.g. after projection

• Multiset relational algebra retains duplicates, to match SQL semantics
  - SQL duplicate retention was initially for efficiency, but is now a feature

• Multiset relational algebra defined as follows
  - selection: has as many duplicates of a tuple as in the input, if the tuple satisfies the selection
  - projection: one tuple per input tuple, even if it is a duplicate
  - cross product: If there are \( m \) copies of \( t_{1} \) in \( r \), and \( n \) copies of \( t_{2} \) in \( s \), there are \( m \times n \) copies of \( t_{1}.t_{2} \) in \( r \times s \)
  - Other operators similarly defined
    - E.g. union: \( m + n \) copies, intersection: \( \min(m, n) \) copies
      - difference: \( \min(0, m - n) \) copies
Relational Algebra

- Those are the basic operations

- What about SQL Joins?
  - Compose multiple operators together
    \[ \sigma_{A=C}(r \times s) \]

- Additional Operations
  - Set intersection
  - Natural join
  - Division
  - Assignment

---

Additional Operators: Join Variations

*Tables: r(A, B), s(B, C)*

<table>
<thead>
<tr>
<th>name</th>
<th>Symbol</th>
<th>SQL Equivalent</th>
<th>RA expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>cross product</td>
<td>(\times)</td>
<td>select * from r, s;</td>
<td>(r \times s)</td>
</tr>
<tr>
<td>natural join</td>
<td>(\Join)</td>
<td>natural join</td>
<td>(\pi_{r.A, r.B, s.C}\sigma_{r.B=s.B}(r \times s))</td>
</tr>
<tr>
<td>equi-join</td>
<td>(\bowtie_\theta)</td>
<td>(theta must be equality)</td>
<td></td>
</tr>
<tr>
<td>theta join</td>
<td>(\bowtie_\theta)</td>
<td>from .. where (\theta);</td>
<td>(\sigma_\theta(r \times s))</td>
</tr>
<tr>
<td>left outer join</td>
<td>(r \bowtie s)</td>
<td>left outer join (with “on”)</td>
<td>(see previous slide)</td>
</tr>
<tr>
<td>full outer join</td>
<td>(r \bowtie s)</td>
<td>full outer join (with “on”)</td>
<td>–</td>
</tr>
<tr>
<td>(left) semijoin</td>
<td>(r \bowtie s)</td>
<td>none</td>
<td>(\pi_{r.A, r.B}(r \bowtie s))</td>
</tr>
<tr>
<td>(left) antijoin</td>
<td>(r \bowtie s)</td>
<td>none</td>
<td>(r - \pi_{r.A, r.B}(r \bowtie s))</td>
</tr>
</tbody>
</table>
Additional Operators

- Set intersection ($\cap$)
  - $r \cap s = r - (r - s)$
  - SQL Equivalent: intersect

- Assignment ($\leftarrow$)
  - A convenient way to write complex RA expressions
  - Essentially for creating “temporary” relations
    - $temp1 \leftarrow \Pi_{R-S}(r)$
  - SQL Equivalent: “create table as...”

Outline

- Relational Algebra (6.1)
- E/R Model (7.2 - 7.4)
- E/R Diagrams (7.5)
- Reduction to Schema (7.6)
- Relational Database Design (7.7)
- Functional Dependencies (8.1 – 8.4)
- Normalization (8.5 – 8.7)
- Formal Semantics of SQL
First, define your *entities*:
- An object that *exists* and is *distinguishable* from other objects
  - Examples: Bob Smith, BofA, CMSC424
- Have *attributes* (people have names and addresses)
- Form *entity sets* with other entities of the same type that share the same properties
  - Set of all people, set of all classes
- Entity sets may overlap
  - Customers and Employees

Second, define the entity *relationships*:
- Relate 2 or more entities
  - E.g. Bob Smith *has account at* College Park Branch
- Form *relationship sets* with other relationships of the same type that share the same properties
  - Customers *have accounts at* Branches
  - “Bob has account 1234” is a *relationship*
  - The set of all mappings from people to accounts is a *relationship set*
- Can have attributes:
  - *has account at* may have an attribute *start-date*
- Can involve more than 2 entities
  - Employee *works at* Branch *at* Job
Next: Relationship Cardinalities

- We may know:
  - One customer can only open one account
  - OR
  - One customer can open multiple accounts

- Representing this is important

- Why?
  - Better manipulation of data
    - If former, can store the account info in the customer table
  - Can enforce such a constraint
    - “Application logic will handle it” NOT GOOD
  - If not represented in conceptual model, domain knowledge can easily be lost

Next: Types of Attributes

- Simple vs Composite
  - Single value per attribute?
    - Are parts accessed separately?
    - e.g. accessing first and last names from name

- Single-valued vs Multi-valued
  - E.g. Phone numbers are multi-valued

- Derived
  - If date-of-birth is present, age can be derived
  - Can help in avoiding redundancy, enforcing constraints etc...
Relational Algebra (6.1)

E/R Model (7.2 - 7.4)

E/R Diagrams (7.5)

Reduction to Schema (7.6)

Relational Database Design (7.7)

Functional Dependencies (8.1 – 8.4)

Normalization (8.5 – 8.7)

Relational Query Languages

SQL Basics

Formal Semantics of SQL

E-R Diagrams

- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Attributes listed inside entity rectangle
- Underline indicates primary key attributes
Composite, Multivalued, and Derived

instructor

<table>
<thead>
<tr>
<th>ID</th>
</tr>
</thead>
<tbody>
<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>street_number</td>
</tr>
<tr>
<td>street_name</td>
</tr>
<tr>
<td>apt_number</td>
</tr>
<tr>
<td>city</td>
</tr>
<tr>
<td>state</td>
</tr>
<tr>
<td>zip</td>
</tr>
<tr>
<td>phone_number</td>
</tr>
<tr>
<td>date_of_birth</td>
</tr>
<tr>
<td>age ( )</td>
</tr>
</tbody>
</table>

Relationship Sets with Attributes

instructor

<table>
<thead>
<tr>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>salary</td>
</tr>
</tbody>
</table>

advisor

| date |

student

<table>
<thead>
<tr>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>tot_cred</td>
</tr>
</tbody>
</table>

instructor student

advisor
Roles

- Entity sets of a relationship need not be distinct
  - Each occurrence of an entity set plays a “role” in the relationship
- The labels “course_id” and “prereq_id” are called roles.

![Diagram of entity relationship]

Participation of an Entity Set in a Relationship Set

- Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set
- E.g., participation of section in sec_course is total
  - every section must have an associated course
- Partial participation: some entities may not participate in any relationship in the relationship set
- Example: participation of course in section is partial

![Diagram of course and section relationship]
Alternative Notation for Cardinality Limits

- Cardinality limits can also express participation constraints

<table>
<thead>
<tr>
<th>Instructor</th>
<th>advisor</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>0..*</td>
<td>ID</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td>name</td>
</tr>
<tr>
<td>salary</td>
<td></td>
<td>tot_cred</td>
</tr>
</tbody>
</table>

What attributes are needed to represent a relationship completely and uniquely?

- Union of primary entities keys, and relationship attributes

<table>
<thead>
<tr>
<th>Instructor</th>
<th>advisor</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td>name</td>
</tr>
<tr>
<td>salary</td>
<td></td>
<td>tot_cred</td>
</tr>
</tbody>
</table>

- \{i.ID, s.ID, date\} describes relationship completely
  - but we may not need them all
Is \{i.ID, s.ID, date\} a candidate key for \textit{advisor}?

- No. Attribute \textit{access-date} can be removed from this set without losing key-ness
- In fact, \textit{union of primary keys of associated entities is always a superkey}

\textit{assuming single advisor and single student only have one advisor relationship}

"a student may have many advisors, and an advisor may advise many students"

\begin{itemize}
  \item No. Primary key on "many" side is enough, i.e. i.ID.
\end{itemize}
Is \{i.ID, s.ID, date\} a candidate key for advisor?

“a student has only a single advisors, and an advisor advises only a single student”

- Either i.ID or s.ID is enough

Assuming single advisor and single student only have one advisor relationship

General rule for binary relationships

- one-to-one: primary key of either entity set
- one-to-many: primary key of the many side
- many-to-many: union of primary keys of the associate entity sets

n-ary relationships

- More complicated rules