Lecture 6:

- Relational Model (Part II: Querying data)

Announcements:

- HW 2 out (due Tues, Sept 26)
- Quiz 3 on Thurs (relational algebra)
**Example Tables**

### Branch

<table>
<thead>
<tr>
<th>branch_name</th>
<th>address</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>906 Main</td>
<td>444-5300</td>
</tr>
<tr>
<td>South Hill</td>
<td>3324 Perry</td>
<td>444-5301</td>
</tr>
</tbody>
</table>

### Account

<table>
<thead>
<tr>
<th>acct_id</th>
<th>acct_name</th>
<th>main_branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Alice</td>
<td>Downtown</td>
</tr>
<tr>
<td>102</td>
<td>Bob</td>
<td>Downtown</td>
</tr>
<tr>
<td>103</td>
<td>Alice</td>
<td>South Hill</td>
</tr>
<tr>
<td>104</td>
<td>Charlie</td>
<td>Downtown</td>
</tr>
</tbody>
</table>

### Loan

<table>
<thead>
<tr>
<th>acct_id</th>
<th>barcode</th>
<th>checkout_date</th>
<th>due_date</th>
<th>return_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>4242</td>
<td>8/12</td>
<td>8/26</td>
<td>8/24</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>8/12</td>
<td>8/19</td>
<td>NULL</td>
</tr>
<tr>
<td>102</td>
<td>4242</td>
<td>8/25</td>
<td>9/7</td>
<td>8/29</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>7/10</td>
<td>7/17</td>
<td>7/18</td>
</tr>
</tbody>
</table>
Relational Algebra Review (From Last Time)

The **selection** \( \sigma_\theta(R) \) of relation \( R \)
- returns a new relation with \( R \) tuples that match the boolean expression \( \theta \)

The **projection** \( \pi_{a_1, a_2, \ldots, a_n}(R) \) of relation \( R \)
- returns a new relation of \( R \) tuples with attributes \( a_1, a_2, \ldots, a_n \) from \( R \)
- *note*: duplicate rows are removed in the returned relation

The **renaming** \( \rho_{S(a'_1, a'_2, \ldots, a'_{n'})(R)} \) of relation \( R \)
- returns a new relation identical to \( R \) but with name \( S \) and attributes \( a'_1 \) to \( a'_{n'} \)
- if only the relation name is changed, can shorten to \( \rho_S(R) \)
- if only a few attributes are changed, can shorten to \( \rho_{a_i \rightarrow a'_i, a_j \rightarrow a'_j}(R) \)

The **cartesian product** (or just *product*) \( R \times S \) of relations \( R \) and \( S \)
- returns a new relation with every \( R \) tuple combined with every \( S \) tuple

The **natural join** \( R \bowtie S \) of relations \( R \) and \( S \)
- a product that pairs tuples whose values match on shared attributes
- returns only uniquely named attributes (removes duplicate attribute names)

The **theta join** \( R \bowtie_\theta S \) of relations \( R \) and \( S \)
- a product that matches \( R \) and \( S \) tuples based on a conditional expression \( \theta \)
- to identify attributes we can prefix with relation names \( R.x, S.u, \) etc.
- can simplify \( R \bowtie_{x=x} S \) to \( R \bowtie_x S \) (for equality of same attributes in \( R, S \))
The classic set operations (union, intersect, difference)

The union \( R \cup S \) of relations \( R \) and \( S \)

- returns a new relation with all tuples in \( R \) and \( S \)
- a tuple only appears once in result even if in both \( R \) and \( S \) (no duplicates)

The intersection \( R \cap S \) of relations \( R \) and \( S \)

- returns a new relation with tuples in both \( R \) and \( S \) ... no duplicates

The difference \( R \setminus S \) of relations \( R \) and \( S \)

- returns a new relation with tuples in \( R \) that are not in \( S \)

To use the basic set operators, relations must be “union compatible”

- \( R \) and \( S \) must have schemas with identical sets of attributes
- and the types (domains) of the matching attributes must be compatible

Example: Assume we separated accounts into tables based on their branch:

\[
\text{DowntownAccount}(\text{acct_id}, \text{acct_name})
\]

\[
\text{SouthHillAccount}(\text{acct_id}, \text{acct_name})
\]

where the same id can occur in different tables ...

Q: What would DowntownAccount \( \cup \) SouthHillAccount be used for?

Q: What would DowntownAccount \( \cap \) SouthHillAccount be used for?

Q: What would DowntownAccount \( \setminus \) SouthHillAccount be used for?
Check In:

Use the following relations to write the relational algebra queries below.

Course(cid, dept, num, credits)
Instructor(iid, name, dept)
Schedule(cid, semester, section, enrollment, iid)

1. Find course information for courses over 3 credits.
   \( \sigma_{\text{credit}>3}(\text{Course}) \)

2. Find courses (cid’s) offered in the Fall of 2023 (F23) with 15 or fewer students.
   \( \pi_{\text{cid}}(\sigma_{\text{semester}=\text{F23} \land \text{enrolled} \leq 15}(\text{Schedule})) \)

3. Modify (2) to also return departments and numbers (dept and num).
   \( \pi_{\text{cid,dept,num}}(\text{Course} \bowtie_{\text{cid}} \sigma_{\text{semester}=\text{F23} \land \text{enrolled} \leq 15}(\text{Schedule})) \)

4. Find instructors that have taught a course in a different department (than the instructor).
   \( \pi_{\text{iid, name, dept}}(\sigma_{\text{Course.dept} \neq \text{Instructor.dept}}(\text{Course} \bowtie_{\text{cid}} (\text{Schedule} \bowtie_{\text{iid}} \text{Instructor}))) \)

5. Find course information for courses that have never been scheduled.
   \( \text{Course} \bowtie_{\text{cid}} (\pi_{\text{cid}}(\text{Course}) \setminus \pi_{\text{cid}}(\text{Schedule})) \)
Query trees

Relational algebra queries, e.g.:

\[ \pi_{\text{main\_branch}}(\text{Account} \bowtie_{\text{acct\_id}=\text{acct\_id}} (\sigma_{\text{return\_date} \geq \text{due\_date}}(\text{Loan}))) \]

are sometimes drawn as query trees ... i.e., as dataflow pipelines

In a query tree

- relations become leaf nodes
- operators are internal, non-leaf nodes
- children nodes provide input to parent nodes
- the output of the root node is the query result

Most DBMSs use something similar internally for query optimization and execution
Some examples of relational algebra equivalences (not comprehensive)

1. \( R \bowtie_C S = S \bowtie_C R \) ...
   commutative, also: \( \cup, \cap, \times \)

2. \( R \cap S = R \setminus (R \setminus S) \)

3. \( R \bowtie_C S = \sigma_C(R \times S) \)

4. \( \sigma_{C_1 \land C_2}(R) = \sigma_{C_2}(\sigma_{C_1}(R)) = \sigma_{C_1}(\sigma_{C_2}(R)) \)

5. \( \sigma_{C_1 \lor C_2}(R) = \sigma_{C_1}(R) \cup \sigma_{C_2}(R) \)

6. \( \sigma_{C_1}(R) \bowtie_C S = \sigma_{C_1}(R \bowtie_C S) \) ...
   if \( C_1 \) mentions only attributes of \( R \)

7. \( \pi_A(\sigma_C(R)) = \sigma_C(\pi_A(R)) \) ...
   if \( C \) only mentions attributes in \( A \)

(*) the 6th equivalence is called “pushing a select”

There are additional operations/extensions we won’t cover

- semi-join (like natural join, but doesn’t remove duplicate columns)
- anti-join (non matches)
- divide \( (R \div S \text{ finds all } R \text{ tuples, w.r.t. non } S \text{ atts, that match all } S \text{ tuples}) \)
- grouping and aggregation
Schema Diagram Basics

1. Each table drawn as a labeled box (with keys designated)

2. Foreign keys as lines between boxes (sometimes linking attributes)

Sometimes called a “table diagram” ... but **not** an “ER diagram”