Announcements (Thurs. September 1)

- Please sign up for mailing list and database (IBM DB2) accounts on the sign-up sheet (now circulating)
- Homework #1 will be assigned next Tuesday
- Office hours: see also course Web page
  - Jun: TTH afternoon
  - Ming: MW late afternoon
- Book update
  - $101 (new) / $75.75 (used) from Duke bookstore
    - Available possibly tomorrow and definitely by next Tuesday
  - $86.15 (new, free shipping) from Amazon

Relational data model

- A database is a collection of relations (or tables)
- Each relation has a list of attributes (or columns)
- Each attribute has a domain (or type)
  - Set-valued attributes not allowed
- Each relation contains a set of tuples (or rows)
  - Each tuple has a value for each attribute of the relation
  - Duplicate tuples are not allowed
    - Two tuples are identical if they agree on all attributes
- Simplicity is a virtue!

Example

```latex
\begin{tabular}{|c|c|c|}
\hline
SID & Name & Age & GPA \\
\hline
142 & Bart & 10 & 2.3 \\
123 & Milhouse & 10 & 3.1 \\
857 & Lisa & 8 & 4.3 \\
456 & Ralph & 8 & 2.3 \\
\hline
\end{tabular}
```

```latex
\begin{tabular}{|c|c|}
\hline
CID & Title \\
\hline
CPS116 & Intro. to Database Systems \\
CPS130 & Analysis of Algorithms \\
CPS114 & Computer Networks \\
\hline
\end{tabular}
```

```latex
\begin{tabular}{|c|c|}
\hline
SID & CID \\
\hline
142 & CPS116 \\
142 & CPS114 \\
123 & CPS116 \\
857 & CPS116 \\
857 & CPS130 \\
456 & CPS114 \\
\hline
\end{tabular}
```

Ordering of rows doesn’t matter (even though the output is always in some order)

Schema versus instance

- Schema (metadata)
  - Specification of how data is to be structured logically
  - Defined at set-up
  - Rarely changes
- Instance
  - Content
  - Changes rapidly, but always conforms to the schema
- Compare to type and objects of type in a programming language

Example

- Schema
  - Student (SID integer, name string, age integer, GPA float)
  - Course (CID string, title string)
  - Enroll (SID integer, CID integer)
- Instance
  - \{ (142, Barr, 10, 2.3), (123, Milhouse, 10, 3.1), ... \}
  - \{ (CPS116, Intro. to Database Systems), ... \}
  - \{ (142, CPS116), (142, CPS114), ... \}
Relational algebra operators

A language for querying relational databases based on operators:

- Core set of operators:
  - Selection, projection, cross product, union, difference, and renaming
- Additional, derived operators:
  - Join, natural join, intersection, etc.
- Compose operators to make complex queries

Selection

- Input: a table \( R \)
- Notation: \( \sigma_p R \)
- \( p \) is called a selection condition/predicate
- Purpose: filter rows according to some criteria
- Output: same columns as \( R \), but only rows of \( R \) that satisfy \( p \)

Selection example

- Students with GPA higher than 3.0
  \[ \sigma_{\text{GPA} > 3.0} \text{Student} \]

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
</tbody>
</table>

More on selection

- Selection predicate in general can include any column of \( R \), constants, comparisons (\( =, \leq, \text{etc.} \)), and Boolean connectives (\( \land: \text{and}, \lor: \text{or}, \text{and} \neg: \text{not} \))
- Example: straight A students under 18 or over 21
  \[ \sigma_{\text{GPA} \geq 4.0 \land (\text{age} < 18 \lor \text{age} > 21)} \text{Student} \]
- But you must be able to evaluate the predicate over a single row of the input table
- Example: student with the highest GPA
  \[ \sigma_{\text{GPA} = \text{all GPA in Student table}} \text{Student} \]

Projection

- Input: a table \( R \)
- Notation: \( \pi_L R \)
- \( L \) is a list of columns in \( R \)
- Purpose: select columns to output
- Output: same rows, but only the columns in \( L \)

Projection example

- ID’s and names of all students
  \( \pi_{\text{SID, name}} \text{Student} \)

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>2.3</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>3.1</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>2.3</td>
</tr>
</tbody>
</table>
More on projection

- Duplicate output rows are removed (by definition)
  - Example: student ages

\[ \pi_{\text{name}} \text{Student} \]

\[ \pi_{\text{age}} \text{Student} \]

Cross product

- Input: two tables \( R \) and \( S \)
- Notation: \( R \times S \)
- Purpose: pairs rows from two tables
- Output: for each row \( r \) in \( R \) and each row \( s \) in \( S \), output a row \( rs \) (concatenation of \( r \) and \( s \))

Cross product example

- \( \text{Student} \times \text{Enroll} \)

A note on column ordering

- The ordering of columns in a table is considered unimportant (as is the ordering of rows)
- That means cross product is commutative, i.e., \( R \times S = S \times R \) for any \( R \) and \( S \)

Derived operator: join

- Input: two tables \( R \) and \( S \)
- Notation: \( R \leftarrow p S \)
  - \( p \) is called a join condition/predicate
- Purpose: relate rows from two tables according to some criteria
- Output: for each row \( r \) in \( R \) and each row \( s \) in \( S \), output a row \( rs \) if \( r \) and \( s \) satisfy \( p \)
- Shorthand for \( \sigma_p (R \times S) \)

Join example

- Info about students, plus CID’s of their courses

Use \text{table_name}.\text{column_name} syntax to disambiguate identically named columns from different input tables
Derived operator: natural join
- Input: two tables $R$ and $S$
- Notation: $R \bowtie S$
- Purpose: relate rows from two tables, and
  - Enforce equality on all common attributes
  - Eliminate one copy of common attributes
- Shorthand for $\pi_{L}(\pi_{p}(R \bowtie S))$, where
  - $p$ equates all attributes common to $R$ and $S$
  - $L$ is the union of all attributes from $R$ and $S$, with
duplicate attributes removed

Natural join example
- $\text{Student} \bowtie \text{Enroll} = \pi_{\text{ID, name, age, GPA, CID}}(\text{Student, SID} = \text{Enroll.SID, Enroll})$

Union
- Input: two tables $R$ and $S$
- Notation: $R \cup S$
  - $R$ and $S$ must have identical schema
- Output:
  - Has the same schema as $R$ and $S$
  - Contains all rows in $R$ and all rows in $S$, with
duplicate rows eliminated

Difference
- Input: two tables $R$ and $S$
- Notation: $R - S$
  - $R$ and $S$ must have identical schema
- Output:
  - Has the same schema as $R$ and $S$
  - Contains all rows in $R$ that are not found in $S$

Derived operator: intersection
- Input: two tables $R$ and $S$
- Notation: $R \cap S$
  - $R$ and $S$ must have identical schema
- Output:
  - Has the same schema as $R$ and $S$
  - Contains all rows that are in both $R$ and $S$
- Shorthand for $R - (R - S)$
- Also equivalent to $S - (S - R)$
- And to $R \setminus S$

Renaming
- Input: a table $R$
- Notation: $\rho_{S}R$, or $\rho_{S(A_{1}, A_{2}, \ldots)}R$
- Purpose: rename a table and/or its columns
- Output: a renamed table with the same rows as $R$
- Used to
  - Avoid confusion caused by identical column names
  - Create identical columns names for natural joins
Renaming example

- SID’s of students who take at least two courses

\[ \pi_{\text{sid}}(\text{Enroll}, \text{Enroll}) \]

Expression tree syntax:

\[ \rho_{\text{Enroll}(\text{sid1}, \text{cid1})} \]

\[ \rho_{\text{Enroll}(\text{sid2}, \text{cid2})} \]

\[ \text{Enroll} \]

Summary of core operators

- Selection: \( \sigma_p R \)
- Projection: \( \pi_L R \)
- Cross product: \( R \times S \)
- Union: \( R \cup S \)
- Difference: \( R - S \)
- Renaming: \( \rho_{A_1, A_2, \ldots} R \)

- Does not really add to processing power

Summary of derived operators

- Join: \( R \bowtie S \)
- Natural join: \( R \bowtie S \)
- Intersection: \( R \cap S \)

- Many more
  - Semi-join, anti-semi-join, quotient, …

An exercise

- Names of students in Lisa’s classes

- Their names \( \pi_{\text{name}} \)

- Students in Lisa’s classes \( \pi_{\text{sid}} \)

- Lisa’s classes \( \pi_{\text{cid}} \)

- Enroll

Who’s Lisa?

\[ \sigma_{\text{name}} = \text{“Lisa”} \]

Another exercise

- CID’s of the courses that Lisa is NOT taking

- All CID’s

- CID’s of the courses that Lisa IS taking

- Course

- Enroll

- \( \sigma_{\text{name}} = \text{“Lisa”} \)

- Student

A trickier exercise

- Who has the highest GPA?
  - Who does NOT have the highest GPA?
  - Whose GPA is lower than somebody else’s?

- \( \pi_{\text{sid}} \)
- \( \pi_{\text{student1.sid}} \)

- Student

- Student

- Student

- Student

- Student

- Student

A deeper question:

When (and why) is “−” needed?
**Monotone operators**

Add more rows to the input...

- If some old output rows may need to be removed
  - Then the operator is non-monotone
- Otherwise the operator is monotone
  - That is, old output rows always remain "correct" when more rows are added to the input
  - Formally, for a monotone operator $RelOp$: $R \subseteq R'$ implies $RelOp(R) \subseteq RelOp(R')$

**Classification of relational operators**

- Selection: $\sigma_p R$
  - Monotone
- Projection: $\pi_L R$
  - Monotone
- Cross product: $R \times S$
  - Monotone
- Join: $R \bowtie S$
  - Monotone
- Natural join: $R \bowtie S$
  - Monotone
- Union: $R \cup S$
  - Monotone
- Difference: $R - S$
  - Monotone w.r.t. $R$; non-monotone w.r.t $S$
- Intersection: $R \cap S$
  - Monotone

**Why is “−” needed for highest GPA?**

- Composition of monotone operators produces a monotone query
  - Old output rows remain "correct" when more rows are added to the input
- Highest-GPA query is non-monotone
  - Current highest GPA is 4.1
  - Add another GPA 4.2
  - Old answer is invalidated
  - So it must use difference!

**Why do we need core operator X?**

- Difference
  - The only non-monotone operator
- Cross product
  - The only operator that adds columns
- Union
  - The only operator that allows you to add rows?
  - A more rigorous argument?
- Selection? Projection?
  - Homework problem ☺

**Why is r.a. a good query language?**

- Simple
  - A small set of core operators who semantics are easy to grasp
- Declarative?
  - Yes, compared with older languages like CODASYL
  - Though operators do look somewhat "procedural"
- Complete?
  - With respect to what?

**Relational calculus**

- $\{ s.SID ~|~ s \in Student ~\land~ \neg(\exists s' \in Student: s.GPA < s'.GPA) \}$, or $\{ s.SID ~|~ s \in Student ~\land~ (\forall s' \in Student: s.GPA \geq s'.GPA) \}$
- Relational algebra = “safe” relational calculus
  - Every query expressible as a safe relational calculus query is also expressible as a relational algebra query
  - And vice versa

**Example of an unsafe relational calculus query**

- $\{ s.name ~|~ \neg(s \in Student) \}$
- Cannot evaluate this query just by looking at the database
Turing machine?

- Relational algebra has no recursion
  - Example of something not expressible in relational algebra: Given relation `Parent(parent, child)`, who are Bart’s ancestors?
- Why not Turing machine?
  - Optimization becomes undecidable
  - You can always implement it at the application level
- Recursion is added to SQL nevertheless!