Recommended Readings

- Chapter (complete book)
- Chapter 4
  - Section 4.3, 4.4
- Chapter 5
  - Section 5.3
- http://philip.greenspun.com/sql/
  - Simple queries

Last Lecture

- What you have learnt
  - Simple SQL queries
  
  SELECT [DISTINCT] c_1, c_2, ..., c_m
  FROM   R_1, R_2, ..., R_n
  [WHERE condition]

  For every combination of tuples in R_1, ..., R_n,
  if that combination satisfies condition (i.e., condition is true for that combination),
  then output c_1, ..., c_m columns from that combination
Last Lecture

• Make a cross product of relations in FROM clause, i.e., $R_1 \times R_2 \times \ldots \times R_n$
• For every tuple $t$ in $R_1 \times \ldots \times R_n$,
  If $t$ satisfies condition,
  Then output $c_1, \ldots, c_m$ columns of $t$

Last Lecture

• Relational Algebra
  – Select
  – Project
  – Cross-Product
  – Union
  – Difference
Today's Lecture

- Additional Operators
- Relational Algebra with Bag semantics
- Relational Calculus

Additional Relational Operators

- The set of operators selection, projection, product, union, difference are the basic relational operators
- A query language is relationally complete if it can express any query expressible in relational algebra using these operators
- Additional relational operators include intersection, joins (conditional join (or θ-join), equijoin, natural join), division, and renaming. They are not essential but useful
Intersection - $R \cap S$

- Returns tuples that exist in both $R$ and $S$
- $R$ and $S$ must be union-compatible
- The output contains attributes identical to $R$ or $S$
- $R \cap S = R - (R - S) = S - (S - R)$

$R$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₂</td>
</tr>
</tbody>
</table>

$S$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
</tr>
<tr>
<td>a₃</td>
<td>b₃</td>
<td>c₃</td>
</tr>
</tbody>
</table>

$R \cap S$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
</tr>
</tbody>
</table>

Venn diagram

- $R - S$
- $R \cap S$
- $S - R$
- $R \cup S$
Conditional Join (or \( \theta \)-join)

- \( R \bowtie_{\theta} S \)
- Meaning: take the cartesian product (or cross-product) of \( R \) and \( S \) and eliminate tuples that do not satisfy the condition \( \theta \)
- Equivalent to writing it as \( \sigma_{\theta} (R \times S) \)
- \( \theta \) is typically an equality condition or a conjunction of equality condition but it can also be others

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_2 )</td>
<td>78</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_1 )</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>( e_2 )</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>( e_3 )</td>
<td>39</td>
<td></td>
</tr>
</tbody>
</table>

\( R \bowtie_{C=D} S \)

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>89</td>
<td>100</td>
<td>( e_1 )</td>
<td></td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_2 )</td>
<td>78</td>
<td>100</td>
<td>( e_1 )</td>
<td></td>
</tr>
</tbody>
</table>

Winter 2003
**Equijoin**

- Similar to conditional join where the condition \( \theta \) contains only conjunctions of equality conditions.
  i.e., \( \theta_1 \) AND \( \theta_2 \) AND … AND \( \theta_n \) where each \( \theta_i \) is an equality condition of the form \( R.\text{name1} = S.\text{name2} \)
- Resulting set of attributes similar to cross-product, but only one copy of fields for which equality is specified is retained (\( R.\text{name1} \) is retained).

<table>
<thead>
<tr>
<th>( R )</th>
<th>( S )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>a_1</td>
<td>b_1</td>
</tr>
<tr>
<td>a_2</td>
<td>b_2</td>
</tr>
</tbody>
</table>

\( R \bowtie_{c \in D} S \)

| \( R \bowtie_{c \in D} S \) | \( A \) | \( B \) | \( C \) | \( E \) |
|---|---|---|---|
| a_2 | b_2 | 78 | \( e_2 \) |

**Natural Join**

- \( R \bowtie S \) (no subscript)
- Equijoin on all common fields in \( R \) and \( S \)
- Example:
  – A and C are the common fields of \( R(A, B, C, D) \) and \( S(A, C, E, F) \)
  – Therefore,
    - \( R \bowtie S = R \bowtie_{R.A = S.A \text{ AND } R.C = S.C} S \)
- The resulting relation has attributes A, B, C, D, E, F
- How do you express natural join with basic relational algebra operators?
Example - Natural Join

- \( R \circ \rho_D \rightarrow_C (S) \)

<table>
<thead>
<tr>
<th>R</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>a_1</td>
<td>b_1</td>
<td>89</td>
</tr>
<tr>
<td>a_2</td>
<td>b_2</td>
<td>78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>100</td>
<td>e_1</td>
</tr>
<tr>
<td>78</td>
<td>e_2</td>
</tr>
<tr>
<td>39</td>
<td>e_3</td>
</tr>
</tbody>
</table>

\( R \circ S \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_2</td>
<td>b_2</td>
<td>78</td>
<td>e_2</td>
</tr>
</tbody>
</table>

Division (Quotient) - \( R/S \) or \( R \div S \)

- In \( R/S \), the set of attributes in \( S \) must be a subset of the attributes in \( R \); every attribute that occurs in \( S \) also occurs in \( R \).
- Let \( R(A_1, \ldots, A_m, B_1, \ldots, B_n) \) and \( S(B_1, \ldots, B_n) \) be the schemas where \( n>0, m>0 \).
- Meaning: Intuitively, if you have relations \( R(A,B) \) and \( S(B) \), the tuple \( <a> \in R/S \) if for every tuple \( <b> \in S \), \( <a,b> \in R \).
Example - Division

\[
R = \begin{array}{ccc}
A & B & C \\
a_1 & b_1 & c_1 \\
a_1 & b_2 & c_2 \\
a_2 & b_1 & c_1 \\
a_1 & b_3 & c_3 \\
a_4 & b_2 & c_2 \\
a_3 & b_2 & c_2 \\
a_4 & b_1 & c_1 \\
\end{array} \\
S = \begin{array}{cc}
B & C \\
b_1 & c_1 \\
b_2 & c_2 \\
\end{array} \\
R/S = \begin{array}{c}
A \\
a_1 \\
a_4 \\
\end{array}
\]

Division

• Another example
  – Enrollments(sid,cid,grade)
  – Courses(cid, cname)
  – Find all students who have taken all courses
    • Enrollments / (π_{cid} (Courses))
Division

- Given $R(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $S(B_1, \ldots, B_n)$, $R/S$ consists of a set of tuples $<a_1, \ldots, a_m>$ where for every tuple $<b_1, \ldots, b_n> \in S$, the tuple $<a_1, \ldots, a_m, b_1, \ldots, b_n>$ exists in $R$

- What happens if $S$ is an empty relation?
- How can we express $R/S$ with basic operators?
  - Show that
    $$R/S = \pi_{A_1, \ldots, A_n}(R) - \pi_{A_1,\ldots, A_n}((\pi_{A_1,\ldots, A_n}(R) \times S) - R)$$

Examples

- Sailors($sid$, sname, rating, age)
- Boats($bid$, bname, color)
- Reserves($sid$, $bid$, day)

- Q1: Find the names of sailors who have reserved boat 103
  - $\pi_{\text{sname}}((\sigma_{\text{bid}=103} \text{ Reserves}) \bowtie \text{ Sailors})$
  - $\pi_{\text{sname}}((\sigma_{\text{bid}=103} \text{ Reserves}) \bowtie \text{ Sailors})$

- Q2: Find colors of boats reserved by Lubber
  - $\pi_{\text{color}}((\sigma_{\text{sname}='Lubber'} \text{ Sailors}) \bowtie \text{ Reserves} \bowtie \text{ Boats})$
  - Is this an equivalent query?
    * $\pi_{\text{color}}((\sigma_{\text{sname}='Lubber'} \text{ Sailors}) \bowtie \text{ Reserves} \bowtie \text{ Boats})$
Examples

- Q3: Find the names of sailors who have reserved at least one boat
  - $\pi_{\text{name}}(\text{Sailors} \bowtie \text{Reserves})$
- Q4: Find the names of sailors age over 20 who have not reserved any boats
  - $\pi_{\text{name}}(\sigma_{\text{age}>20}\text{Sailors}) - Q3$
- Q5: Find the names of sailors who have reserved a red or green boat
  - $\pi_{\text{name}}(\sigma_{\text{color}="\text{red}" \text{ OR color} = "\text{green}"} (\text{Boat})) \bowtie \text{Sailors} \bowtie \text{Reserves}$
  - $\pi_{\text{name}}(\sigma_{\text{color}="\text{red}"} (\text{Boat}) \cup \sigma_{\text{color}="\text{green}"} (\text{Boat})) \bowtie \text{Sailors} \bowtie \text{Reserves}$

Examples

- Q6: Find the names of sailors who have reserved a red and a green boat.
  - $\pi_{\text{name}}(\sigma_{\text{color}="\text{red}" \text{ AND color} = "\text{green}"} (\text{Boat})) \bowtie \text{Sailors} \bowtie \text{Reserves}$
    - Wrong! Empty result always!
  - $Q_{\text{red}} = \pi_{\text{sid}} (\sigma_{\text{color}="\text{red}"} (\text{Boats}) \bowtie \text{Reserves})$
  - $Q_{\text{green}} = \pi_{\text{sid}} (\sigma_{\text{color}="\text{green}"} (\text{Boats}) \bowtie \text{Reserves})$
  - $\pi_{\text{name}} (Q_{\text{red}} \bowtie Q_{\text{green}} \bowtie \text{Sailors})$
- Does the following query compute the same answer? Why?
  - $\pi_{\text{name}} ((Q_{\text{red}} \cap Q_{\text{green}}) \bowtie \text{Sailors})$
Examples

• Q7: Find the names of sailors who have reserved at least 2 boats
  \[ Q = \rho_{\text{sid.1} \rightarrow \text{sid1}, \text{sid.2} \rightarrow \text{sid2}, \text{bid.1} \rightarrow \text{bid1}, \text{bid.2} \rightarrow \text{bid2}} (\text{Reserves} \times \text{Reserves}) \]
  \[ \pi_{\text{sname}} (\sigma_{\text{sid1} = \text{sid2} \land \text{bid1} \neq \text{bid2}} (Q) \bowtie \text{Sailors}) \]

• Can you write a query that finds the names of sailors who have reserved exactly two boats?

Things to ponder

• When is \( R \bowtie S = R \cap S \)?
• When is \( R \bowtie S = R \times S \)?
• When is \( R \bowtie S \subseteq R \)?
• Given that the queries are equivalent? Which is better in terms of efficiency?
  - \( \sigma_{\text{age} > 20} (\text{Sailors} \bowtie \text{Reserves}) \)
  - \( \sigma_{\text{age} > 20} (\text{Sailors}) \bowtie \text{Reserves} \)
• How can a basic SQL query be translated into a relational algebra query and how can a relational algebra query consisting of select, project, and product operators be translated into SQL?
• Is every relational operator in \( \sigma, \pi, \times, \cup, \cap \) necessary? Can one be expressed in terms of the rest?
Summary

• Relational algebra is not as generic and powerful as general purpose programming languages
  – Can we write a recursive query in relational algebra?
• Not intended for calculations that are too complex
• However, the query language is good enough for writing most practical queries and nice properties exist for relational algebra
  – optimization rules
  – operators can be efficiently implemented
• SQL is often translated into possibly many different but equivalent relational algebra queries. The optimizer picks the most efficient one to execute
• A good balance between efficiency and expressiveness

Relational Algebra with Bag Semantics

• Recall
  SELECT age
  FROM  Students

  returns the result { 21, 21, 20, 19 }

• The evaluation engine often does not remove duplicates unless it is explicitly instructed to do so through the DISTINCT keyword
• Why do we use bag semantics?
  – Removing duplicates requires a sort followed by a scan through the sorted results
  – Sometimes it is necessary to retain the multiset result for certain calculations. E.g., find the total age of all students
Relational Algebra with Bag Semantics

- Select, Project
  - Duplicates are not eliminated
  - A tuple that occurs n times in input occurs n times in the output
- Product: RxS
  - Suppose <x₁, ..., xₚ> occurs m times in R and a tuple <y₁, ..., yₚ> occurs n times in S, then <x₁, ..., xₚ, y₁, ..., yₚ> occurs m x n times in RxS
- Union: R ∪ S
  - Duplicates are not eliminated
  - If the sample tuple occurs m times in R and n times in S, then it occurs m+n times in R ∪ S
- Difference: R – S
  - Duplicates are not eliminated
  - Suppose a tuple t occurs m times in R and n times in S, then
    - t occurs max(0, m-n) times in R - S

Relational Calculus

- Relational algebra – an operational formalism for expressing queries
- Relational calculus is another formalism for the relational model
  - Describes the desired result without specifying how to compute it
  - a declarative language
  - Logical formalism for expressing queries based on first-order formulas that describes the desired result
- Big influence on the design of languages such as SQL or QBE (Query by Example)
- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)
Propositional Logic (Quick review)

Recall Propositional Logic:
- An infinite set of propositional variables: \( x, y, z, \ldots \)
- Propositional constants: \( \text{true}, \text{false} \)
- Connectives: \( \neg, \lor, \land, \rightarrow, \leftrightarrow \)
- Variables are assigned with truth values through a truth assignment function \( \phi \)
- Well-Formed Formulas: \( x \land z, (x \lor z \land (\neg y)) \)
- Truth value defined on the structure of the formula

Predicate Logic (Quick review)

- More general than propositional logic
- The syntax of predicate logic starts with variables, constants and predicates that can be built using a collection of boolean-valued operators (boolean expressions)
  - Constant symbols: E.g., Mary, 34, true, ...
  - Variables: \( x, y, z \ldots \)
  - Predicate Symbols.
    - E.g., Person
      - maps tuples of elements to true or false
        - Person(12-33, John, 34)
  - Variables range over elements in a universe using quantifiers \( \exists \) and \( \forall \)
Predicate Logic (Quick review)

- Atoms. (evaluates to true or false)
  - n-ary predicate of n terms E.g. Person(12-33, John, 34), Person(x, y, z)
  - If P and Q are atoms, so is \( \neg P, P \lor Q, P \land Q, P \rightarrow Q, P \leftarrow Q \)
- Sentence: atom or \( (\exists x \, P) \) or \( (\forall x \, P) \) where P is a sentence
- Well-formed formulas: a sentence that does not contain any “free” variable (i.e., all variables are “bound”)
  - \( \exists x \, P(x, y) \) vs \( \exists x \, \forall y \, P(x, y) \)

Tuple Relational Calculus

- Assume the following binary predicates are available and denote \( op \) as one of the following predicates:
  - \( <, =, >, \leq, \neq, \geq \)
- Atomic formula:
  - \( x \in Rel \) where \( x \) is a tuple variable, \( x.a \, op \, y.b \), \( x.a \, op \, constant \)
  - or \( constant \, op \, x.a \)
- Formula: (P and Q are formulas, P(x) is a formula)
  - An atomic formula
  - \( \neg P, P \lor Q, P \land Q, P \rightarrow Q \)
  - \( \exists x(P(x)) \) where \( x \) is a tuple variable
    - “there exists a tuple \( t \) which \( x \) binds to such that \( P(t) \) holds”
  - \( \forall x(P(x)) \) where \( x \) is a tuple variable
    - “for every tuple \( t \) which \( x \) binds to, \( P(t) \) holds”
Semantics

• A TRC query has the form \( \{ x | P(x) \} \)
  – This query is well-formed when \( x \) is the only free variable that occurs in the formula \( P \)
• Semantics:
  – A TRC query \( \{ x | P(x) \} \) consists of all tuples \( t \) that satisfy the TRC formula \( P(t) \)

Examples

Sailors (\( \text{sid}: \) integer, \( \text{sname}: \) string, \( \text{rating}: \) integer, \( \text{age}: \) real)
Boats (\( \text{bid}: \) integer, \( \text{bname}: \) string, \( \text{color}: \) string)
Reserves (\( \text{sid}: \) integer, \( \text{bid}: \) integer, \( \text{day}: \) date)

• \( \{ y | \exists x \in \text{Sailors}, x.\text{age} > 30 \ (y.\text{name} = x.\text{name} \land y.\text{rating} = x.\text{rating}) \} \)
  – The names and ratings of sailors whose age is greater than 30
  – \( y \) is the only free variable, \( x \) is a bound variable
  – For every tuple \( t \) in Sailors, bind \( x \) to \( t \),
    if the age of \( x \) is greater than 30
      return \( y = \langle \text{name}: x.\text{name}, \text{rating}: x.\text{rating} \rangle \)
  – Bind \( y \) to a tuple \( t \) (with name and rating columns)
    if the statement \( P(t) \) is true, return \( y \)
Examples

- \{ z \mid \exists x \in \text{Sailors} \exists y \in \text{Reserves} (x.\text{id}=y.\text{id} \land y.\text{bid}=103 \land z.\text{sname} = x.\text{sname}) \}
  - Find the names of sailors who have reserved boat 103
- \{ w \mid \exists x \in \text{Sailors} \ \forall y \in \text{Boats} \ \exists z \in \text{Reserves} \\
  x.\text{id}=z.\text{id} \land z.\text{bid}=y.\text{bid} \land w.\text{name} = x.\text{name} \}
  - What does the above query compute?
  - Find the names of sailors who have reserved all boats

Examples

- Swap the order:
  - \{ w \mid \exists x \in \text{Sailors} \exists z \in \text{Reserves} \ \forall y \in \text{Boats} \\
  x.\text{id}=z.\text{id} \land z.\text{bid}=y.\text{bid} \land w.\text{name} = x.\text{name} \}
  - What does this query mean? If there is more than one \text{Boat} in \text{Boats} relation, the result of the query is always empty
- \{ w \mid \exists x \in \text{Sailors} \ \forall y \in \text{Boats} \\
  (y.\text{color}="\text{red}" \rightarrow \exists z \in \text{Reserves} x.\text{id}=z.\text{id} \land z.\text{bid}=y.\text{bid} \land x.\text{name}=w.\text{name}) \}
  - Find the names of sailors who have reserved all red boats
Domain Relational Calculus

- Instead of tuple variables, we have domain variables.
- A domain variable ranges over the elements in the domain of some attribute.
- A DRC query has the form
  \( \{ <x_1, \ldots, x_n> \mid P(x_1, \ldots, x_n) \} \)
  
  - \( x_i \) is either a domain variable or a constant
  - \( P(x_1, \ldots, x_n) \) denotes a DRC formula
  - The only free variables are \( x_1, \ldots, x_n \)
- Semantics: The set of all \(<a_1, \ldots, a_n>\) tuples for \( P(a_1, \ldots, a_n) \) evaluates to true (\( a_i \)'s are constants)

DRC formula

- \(<x_1, \ldots, x_n> \in Rel\) where \( Rel \) is a relation with \( n \) attributes and \( x_i \) is either a variable or a constant
- \( x \ op y \)
- \( x \ op \) constant, or constant \ op \( y \)
- A DRC formula is
  - An atomic formula
  - \( \neg P, P \lor Q, P \land Q, P \rightarrow Q \)
  - \( \exists x(P(x)) \) where \( x \) is a domain variable
    - “there exists a value \( c \) which \( x \) binds to such that \( P(c) \) holds”
  - \( \forall x(P(x)) \) where \( x \) is a domain variable
    - “for every value \( c \) which \( x \) binds to, \( P(c) \) holds”
Examples

- Find all sailors older than 30.
  - \{ \langle s, n, r, a \rangle | \langle s, n, r, a \rangle \in \text{Sailors} \land a > 30 \}

- Find the names of all sailors older than 30
  - \{ \langle n \rangle | \exists s, r, a \langle s, n, r, a \rangle \in \text{Sailors} \land a > 30 \}

- Find the names of sailors who have reserved boat 103
  - \{ \langle n \rangle | \exists s, r, a \langle s, n, r, a \rangle \in \text{Sailors} \land \exists b, d \langle b, d \rangle \in \text{Reserves} \land s = s' \land b = 103 \}
  - \{ \langle n \rangle | \exists s, r, a \langle s, n, r, a \rangle \in \text{Sailors} \land \exists d \langle s, 103, d \rangle \in \text{Reserves} \}

- Find the sailors who have reserved all red boats
  - \{ n | \exists s, r, a \langle s, n, r, a \rangle \in \text{Sailors} \land \forall b, b', c \langle b, b', c \rangle \in \text{Boats} \land (c = \text{"red"} \rightarrow \exists s, b, d \langle s, b, d \rangle \in \text{Reserves} \land s = s' \land b = b') \}

Safety

\{ \langle s, n, r, a \rangle | \neg (\langle s, n, r, a \rangle \in \text{Sailors}) \}

- The set of all tuples that are not in the Sailors relation. **Infinite!** (in the context of infinite domains, e.g., Age)
- A query is safe if it produces a finite answer under all possible instances of the input relations
- A program that evaluates an unsafe query will not terminate
- Consider only safe queries.