RA to DRC

Every relational algebra query can be translated into a domain relational calculus query.

• Proof by induction on the structure of the relational algebra query
• In class

DRC to TRC

• Every DRC query can be translated into a tuple relational calculus query.
• Proof by induction on the structure of the DRC query.
• In class
**TRC to RA**

- A safe TRC query can be translated into a RA query
- Therefore, we have the following picture so far

\[ \text{Relational Algebra} \quad \xrightarrow{\text{safe}} \quad \text{Tuple Relational Calculus} \]

\[ \text{Domain Relational Calculus} \]

- More precisely, RA $\equiv$ safe DRC $\equiv$ safe TRC

**Big Deal?**

Relational Algebra and Relational Calculus greatly influenced the design of the first relational databases

- SQL, a relationally complete language, bears strong resemblance to TRC.
  - SELECT x FROM R r WHERE C roughly corresponds to \( \{ x | \exists r \in R, C \} \)

- Relational Algebra:
  - used for optimization of SQL queries. Many equivalent RA queries for the same SQL query. Specialized and efficient algorithms for various relational operators. The relational optimizer picks the best among the possible plans.
  - Performance of relational engines are hard to beat!
Big Deal?

• Relational Calculus:
  – tells us what SQL can or cannot express. E.g., Known that FO logic cannot express recursive queries. E.g. compute the ancestor relation of relation PC(parent, child).
  – To write more complex queries such as recursive queries, extensions are added to SQL. Make a DB talk to host languages etc. The price you pay: your query may not be easily optimizable.

Back to SQL

• Previous lectures
  – Data Manipulation Language (DML)
    • Insert, delete, modify rows, pose queries
  – Data Definition Language (DDL)
    • Create, delete, modify definitions for tables and views
    • Integrity constraints

• We will see other aspects of SQL
  – More advanced constructs of SQL
  – Triggers and Advanced Integrity Constraints
  – Embedded and Dynamic SQL

Tentative
Basic form of SQL

- We have already seen the basic form of SQL:

  \[
  \text{SELECT [DISTINCT] target-list} \\
  \text{FROM relation-list} \\
  [\text{WHERE qualification}]
  \]

  - \text{relation-list}: A list of relation names (possibly with a range-variable or tuple variable after each name). E.g., FROM Sailors s, Reserves r
  - \text{target-list}: A list of attributes of relations in relation-list. * can be used to denote all attributes. E.g., SELECT s.sname, SELECT *
  - \text{qualification}: Comparisons (Attr \text{ op } const or Attr1 \text{ op } Attr2, where \text{ op } is one of $<$, $>$, $\leq$, $\geq$, $\neq$ combined using AND, OR and NOT.
  - \text{DISTINCT} (optional) keyword indicates that the answer should not contain duplicates. Default is that duplicates are not eliminated!

Semantics of the SQL query

- Informally:
  - Result1: Compute the cartesian product of all relations in the relation list
  - Result2: For every tuple in Result1, apply the condition stated in the qualification list. Eliminate the tuple if the condition is not satisfied.
  - Result3: Retrieve only the required components from each tuple in Result2 according to target list
  - Result: Eliminate duplicate rows in Result3 if DISTINCT keyword is present in the SELECT clause
Example Database

<table>
<thead>
<tr>
<th>sid</th>
<th>cname</th>
<th>level</th>
<th>type</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>2336</td>
<td>Cho</td>
<td>Beginner</td>
<td>snowboard</td>
<td>18</td>
</tr>
<tr>
<td>2334</td>
<td>Luke</td>
<td>Inter</td>
<td>snowboard</td>
<td>25</td>
</tr>
<tr>
<td>1887</td>
<td>Ice</td>
<td>Advanced</td>
<td>ski</td>
<td>20</td>
</tr>
<tr>
<td>2339</td>
<td>Paul</td>
<td>Beginner</td>
<td>ski</td>
<td>33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>slope-id</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>2336</td>
<td>s3</td>
<td>01/05/03</td>
</tr>
<tr>
<td>2336</td>
<td>s1</td>
<td>01/06/03</td>
</tr>
<tr>
<td>2336</td>
<td>s1</td>
<td>01/07/03</td>
</tr>
<tr>
<td>1887</td>
<td>s2</td>
<td>01/07/03</td>
</tr>
<tr>
<td>1887</td>
<td>s1</td>
<td>01/07/03</td>
</tr>
<tr>
<td>2334</td>
<td>s2</td>
<td>01/05/03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slope-id</th>
<th>name</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Mountain Run</td>
<td>blue</td>
</tr>
<tr>
<td>s2</td>
<td>Olympic Lady</td>
<td>black</td>
</tr>
<tr>
<td>s3</td>
<td>Magic Carpet</td>
<td>green</td>
</tr>
</tbody>
</table>

Basic SQL Examples

Q1: Find all snowboarders

```
SELECT * FROM Customers c WHERE c.type = "snowboard"
```

<table>
<thead>
<tr>
<th>sid</th>
<th>cname</th>
<th>level</th>
<th>type</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>2336</td>
<td>Cho</td>
<td>Beginner</td>
<td>snowboard</td>
<td>18</td>
</tr>
<tr>
<td>2334</td>
<td>Luke</td>
<td>Inter</td>
<td>snowboard</td>
<td>25</td>
</tr>
</tbody>
</table>

Q2: Find the names of all customers on Mountain Run slope on the day 01/07/03

```
SELECT name FROM Customer c, Activities a, Slopes s WHERE c.sid = a.sid AND s.slope-id = a.slope-id AND a.day="01/07/03" AND s.name = "Mountain Run"
```
DISTINCT

Q3: Return the names of customers who are on the slope for at least one day
SELECT c.sname
FROM Customers c, Activities a
WHERE c.sid=a.sid

SELECT DISTINCT c.cid
FROM Customers c, Activities a
WHERE c.sid=a.sid

UNION

• Return the names of customers who either went on some slope on 11/05/03 or skied on Mountain Run
SELECT c.cname
FROM Customers c, Activities a
WHERE c.sid=a.sid AND a.day="11/05/03"
UNION
SELECT c.cname
FROM Customers c, Activities a, Slopes s
WHERE c.sid=a.sid AND s.slope-id=a.slop-id AND s.sname="Mountain Run"

• The subqueries must be union-compatible
• Note that duplicates are eliminated even though there is no keyword DISTINCT!
• UNION has set semantics.
UNION ALL

SELECT c.cname
FROM Customers c, Activities a
WHERE c.sid=a.sid AND a.day="11/05/03"
UNION ALL
SELECT c.cname
FROM Customers c, Activities a, Slopes s
WHERE c.sid=a.sid AND
  s.slope-id=a.slop-id AND
  s.sname="Mountain Run"

• Duplicates are not eliminated
• UNION ALL has multiset or bag semantics

UNION ALL

• What about the following query?
SELECT DISTINCT c.cname
FROM Customers c, Activities a
WHERE c.sid=a.sid AND a.day="11/05/03"
UNION ALL
SELECT DISTINCT c.cname
FROM Customers c, Activities a, Slopes s
WHERE c.sid=a.sid AND
  s.slope-id=a.slop-id AND
  s.sname="Mountain Run"

• INTERSECT and INTERSECT ALL
EXCEPT

• Set difference
• Find the names of all customers who have been to Mountain Run but not Olympic Lady

```
SELECT c.cname
FROM Customers c, Activities a, Slopes s
WHERE c.sid=a.sid AND s.slope-id=a.slop-id
AND s.sname="Mountain Run"
EXCEPT
SELECT c.cname
FROM Customers c, Activities a, Slopes s
WHERE c.sid=a.sid AND s.slope-id=a.slop-id
AND s.sname="Olympic Lady"
```

• The subqueries must be union-compatible
• Like before, duplicates are eliminated!
• EXCEPT has set-semantics
• EXCEPT ALL will return the name “Cho” twice

Summary

• SELECT ... FROM ... WHERE
• UNION, UNION ALL, INTERSECT, INTERSECT ALL, EXCEPT, EXCEPT ALL
• Can you translate the SQL queries with set semantics shown in this lecture into relational algebra queries?