Today's Lecture

- More on SQL
  - Nested Queries
  - Aggregate operators and Nulls

Recommended Readings

- Chapter 5
  - Section 5.4 - 5.6
- http://philip.greenspun.com/sql/
  - Simple queries, more complex queries
Basic form of SQL

• We have already seen the basic form of SQL:

```
SELECT [DISTINCT] target-list
FROM relation-list
[WHERE qualification]
```

• `relation-list`: A list of relation names (possibly with a range-variable after each name). E.g., FROM Sailors s, Reserves r
• `target-list`: A list of attributes of relations in `relation-list`. `*` can be used to denote all attributes. E.g., SELECT s.sname, SELECT *
• `qualification`: Comparisons (Attr `op` const or Attr1 `op` Attr2, where `op` is one of `<`, `>`, `=`, `≤`, `≥`, ≠) combined using AND, OR and NOT.
• `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: Apply the conditions in qualification list to eliminate tuples from Result1 that do not satisfy the conditions
  – 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
SQL and Relational Algebra

- Indeed, the procedure to arrive at the answer of an SQL query can be expressed in relational algebra

$$\text{SELECT DISTINCT } A_1, \ldots, A_n \quad \pi_{A_1, \ldots, A_n}(\sigma_{\text{condition}}(R_1 \times \ldots \times R_m))$$

Indeed, the procedure to arrive at the answer of an SQL query can be expressed in relational algebra:

$$\text{SELECT DISTINCT } A_1, \ldots, A_n \quad \pi_{A_1, \ldots, A_n}(\sigma_{\text{condition}}(R_1 \times \ldots \times R_m))$$

Example Database

<table>
<thead>
<tr>
<th>sid</th>
<th>name</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>

Slopes

<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>
<tr>
<td>s4</td>
<td>KT-22</td>
<td>black</td>
</tr>
</tbody>
</table>
Expressions and Strings in the SELECT clause

- General form of SELECT clause
  - SELECT expr₁ AS col-name₁, …, exprₙ AS col-nameₙ
  - Expr is any arithmetic or string expressions over column names and constants
  - Col-name is a new name for the corresponding output column according to expr

Examples

SELECT sid, cname, age+1
FROM Customers

SELECT sid AS id, cname AS name, (age*100 – sid)/33 AS shoe-size
FROM Customers
String matching

- Support for string matching through LIKE clause
- % denotes zero or more arbitrary characters
- _ denotes exactly one arbitrary character
- Examples
  - ‘abc’ LIKE a__% is true
  - ‘abc’ LIKE %_%c is true
  - ‘abc’ LIKE a%b%_c is false

- Find the ids of all customers whose name ends with the letter ‘e’
  SELECT sid
  FROM Customers
  WHERE name LIKE ‘%e’

Nested Queries

- A nested query is a query with another query embedded in it
- Embedded query = subquery
- A subquery can in turn have subqueries
- Nested queries are convenient for computing results that depend on some intermediate results that need to be computed
- In SQL, a subquery can appear in the FROM or WHERE or HAVING clause
Examples

- Find the names of customers who went on the slope on 01/07/03

```sql
SELECT cname
FROM Customers c, Activities a
WHERE a.day = '01/07/03' AND a.slope-id = c.sid
```

- Find the names of customers who did not go on the slope on 01/07/03

```sql
SELECT cname
FROM Customers c
WHERE c.sid NOT IN (SELECT a.sid
FROM Activities a
WHERE a.day = '01/07/03')
```
Meaning of Nested Query in WHERE clause

- Result1: Compute the cartesian product of all relations in the relation list in outer query
- Result2: For every tuple in Result1, apply the condition stated in the qualification list. ((Re)compute the subquery in order to apply the condition 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

Correlated Nested Queries

- In all the examples so far, the inner query is independent of the outer query
- The inner query could also depend on the outer query
- Example: Find the names of customers who have been on the slope on 01/07/03

```
SELECT cname
FROM Customers c
WHERE EXISTS (SELECT *
FROM Activities a
WHERE a.sid = c.sid AND a.day="01/07/03")
```

Correlation via range variable c
Checks that the subquery returns a non-empty result
Another example

- Find all customers who went on the slope “Olympic Lady”

```sql
SELECT sid
FROM Customers c
WHERE EXISTS (SELECT a.sid
FROM Activities a
WHERE a.sid = c.sid AND
  a.slope-id IN (SELECT slope-id
  FROM Slopes s
  WHERE s.name='Olympic Lady'))
```

Set comparison operators

- IN, EXISTS, UNIQUE, \( op \) ANY (or \( op \) SOME), \( op \) ALL where \( op \) is one of \{\(<, \leq, =, \leq, \geq, >\}\)

- \( x \) IN \( Q \)
  - Returns true if \( x \) occurs in the set \( Q \)
  - \( x \) NOT IN \( Q \)
    - Returns true if \( x \) does not occur in \( Q \)

- EXISTS \( Q \)
  - Returns true if \( Q \) is a non-empty set
  - NOT EXISTS \( Q \)
    - Returns true if \( Q \) is empty

- What does the following mean?
  - NOT EXISTS (\( B \) EXCEPT \( A \))
Set comparison operators

- **UNIQUE Q**
  - Returns true if every row in Q does not appear more than once
  - If Q is empty?
  - **NOT UNIQUE Q**
    - Returns true if there exists a row in Q that appears more than once

- Find the names of all customers who went to the slopes at most once
  SELECT cname
  FROM Customers c
  WHERE UNIQUE (SELECT c.sid
                  FROM Activities a
                  WHERE c.sid = a.sid)

  Answer: Paul and Luke

Set comparison operators

- **op ANY (or op SOME), op ALL where op is one of \(<, \leq, =, \leq, \geq, >\)**
- **x op SOME Q**
  - Returns true if there exists y in Q such that x op y is true
  - Existential quantification
- **x op ALL Q**
  - Returns true if for every y in Q, x op y is true
  - Universal quantification

Examples:
- Find the names of all customers who went on some slopes
  SELECT c.cname
  FROM Customers c
  WHERE c.sid = SOME (SELECT sid
                        FROM Activities a)
More examples

• Find the names of all skiers whose age is greater than every snowboarder
  SELECT c.name
  FROM Customers c
  WHERE c.type="skier" AND c.age > (SELECT c.age
  FROM Customers c
  WHERE c.type = 'snowboard')

• Find the name of the oldest customers
  SELECT cname
  FROM Customer c
  WHERE c.age >= (SELECT c.age
  FROM Customers c)

Aggregates

• Compute summary results over a table. E.g.,
  – find the average/min/max score of all students who took CS180
  – find the total number of snowboarders
  – find total salary of employees in Sales department

• SQL:
  – COUNT, SUM, AVG, MIN, MAX
COUNT

- Find the total number of customers
  SELECT COUNT(sid)
  FROM Customers

- Find the total number of days of operation
  SELECT COUNT(DISTINCT(day))
  FROM Activities

  Answer: 3

  SELECT COUNT(DISTINCT(day))
  FROM Activities

  Answer: 6

  • Alternatively, the last query could have been written as
    SELECT COUNT(*)
    FROM Activities

SUM, AVG

- SUM(_) and SUM(DISTINCT(_)) have similar obvious meanings
- Find the total profit of the company
  SELECT SUM(qty*price)
  FROM Sales

- Similarly for AVG(_) and AVG(DISTINCT(_))
**MIN, MAX**

• Find the name and age of the oldest snowboarder
  
  ```sql
  SELECT cname, MAX(age)
  FROM Customers c
  WHERE c.type='snowboard'
  
  WRONG!
  ```

• The combination of aggregate and non-aggregate output for the
  SELECT clause is not allowed unless the query also contains a
  GROUP BY clause

---

**MIN, MAX**

• Find the name and age of the oldest snowboarder
  
  ```sql
  SELECT cname, age
  FROM Customers c
  WHERE age = (SELECT MAX(age)
  FROM Customers)
  
  SQL allows this even though the query, rightfully, does not type-check!
  ```
On a similar note...

• Find the activities of Luke

```sql
SELECT *
FROM Activities a
WHERE a.sid = (SELECT sid
               FROM Customers c
               WHERE cname='Luke')
```

• If there is only one Luke in the Customers table, the subquery returns only one sid value. SQL returns that single sid value to be compared with a.sid
• However, if the subquery returns more than one value, a run-time error occurs

More Examples

• Find the names of all skiers whose age is greater than every snowboarder

```sql
SELECT c.name
FROM Customers c
WHERE c.age >ALL (SELECT c.age
                   FROM Customers c
                   WHERE c.type = 'snowboard')
```

```sql
SELECT c.name
FROM Customers c
WHERE c.age > (SELECT MAX(c.age)
               FROM Customers c
               WHERE c.type = 'snowboard')
```
More Examples

• Find the names of all skiers whose age is greater than some snowboarder
  SELECT c.name
  FROM Customers c
  WHERE c.age > SOME (SELECT c.age
                        FROM Customers c
                        WHERE c.type = 'snowboard')

SELECT c.name
FROM Customers c
WHERE c.age > (SELECT MIN(c.age)
                FROM Customers c
                WHERE c.type = 'snowboard')

ORDER BY

SELECT ...
FROM ...
WHERE ...
ORDER BY c₁, ..., cₙ

• Sorts the result according to columns c₁, ..., cₙ
• By default, it sorts the result in ascending order
• Specify descending through ORDER BY c₁, ..., cₙ DESC
GROUP BY and HAVING

- So far the aggregate operations have been applied to all qualifying tuples
- We also want to apply aggregate operations to different groups of tuples. E.g.,
  - find the total regional sales
  - Find the average age of customers in each category

Find the average age of customers in each category

<table>
<thead>
<tr>
<th>type</th>
<th>Avg-age</th>
</tr>
</thead>
<tbody>
<tr>
<td>snowboard</td>
<td>22</td>
</tr>
<tr>
<td>ski</td>
<td>27</td>
</tr>
</tbody>
</table>
GROUP BY and HAVING

SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
GROUP BY grouping-list
HAVING group-qualification

• More specifically, the target list consists of
  – a list of column names
  – and a list of terms having the form
    agg(column-name) AS new-name
• Every column in the list of column names must appear in grouping-list
• The expressions appearing in group-qualification must have a single value
  per group. HAVING states the condition that the resulting tuple (for a
  group) must satisfy before it can be emitted.
• If GROUP BY is omitted, the entire table is regarded as a group

Semantics

• Compute the cartesian product of all relations in the relation list in
  outer query
• For every tuple in Result1, apply the condition stated in the
  qualification list. ((Re)compute the subquery in order to apply the
  condition in the qualification list) Eliminate the tuple if the condition
  is not satisfied.
• Keep only the required columns: columns in the SELECT clause,
  GROUP BY, and HAVING clauses are kept
• Sort the table according to the GROUP BY columns
• Apply group-qualification to each group (according to the GROUP
  BY columns). Eliminate groups which do not satisfy the group-
  qualification
• Generate one tuple for each group according to SELECT clause
Example

- Find the age of the youngest sailor with age $\geq 18$, for each rating with at least 2 such sailors

```sql
SELECT S.rating, MIN(S.age) FROM Sailors S WHERE S.age >= 18 GROUP BY S.rating HAVING COUNT(*) > 1
```

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>92</td>
<td>Frodo</td>
<td>1</td>
<td>28.0</td>
</tr>
<tr>
<td>38</td>
<td>Sam</td>
<td>1</td>
<td>30.0</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Example

- Take the cross product of all relations in the FROM clause

```sql
SELECT S.rating, MIN(S.age) FROM Sailors S WHERE S.age >= 18 GROUP BY S.rating HAVING COUNT(*) > 1
```

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>92</td>
<td>Frodo</td>
<td>1</td>
<td>28.0</td>
</tr>
<tr>
<td>38</td>
<td>Sam</td>
<td>1</td>
<td>30.0</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Example

• Apply the condition in the WHERE clause to every tuple

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>92</td>
<td>Frodo</td>
<td>1</td>
<td>28.0</td>
</tr>
<tr>
<td>38</td>
<td>Sam</td>
<td>1</td>
<td>30.0</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Example

• Keep only the required columns: columns in the SELECT clause, GROUP BY, and HAVING clauses are kept

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>92</td>
<td>Frodo</td>
<td>1</td>
<td>28.0</td>
</tr>
<tr>
<td>38</td>
<td>Sam</td>
<td>1</td>
<td>30.0</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Example

- Sort the table according to the GROUP BY columns

```
SELECT S.rating, MIN(S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) > 1
```

Example

- Apply group-qualification to each group according to the GROUP BY columns. Eliminate groups which do not satisfy the group-qualification

```
SELECT S.rating, MIN(S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) > 1
```
Example

- Generate one tuple for each group according to SELECT clause

```
SELECT S.rating, MIN(S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) > 1
```

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28.0</td>
</tr>
<tr>
<td>1</td>
<td>30.0</td>
</tr>
<tr>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>8</td>
<td>55.5</td>
</tr>
</tbody>
</table>

EVERY and ANY in HAVING

```
SELECT S.rating, MIN(S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) > 1 AND EVERY (S.age ≤ 40)
```

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28.0</td>
</tr>
</tbody>
</table>
EVERY and ANY in HAVING

```sql
SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1 AND SOME (S.age > 40)
```

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28.0</td>
</tr>
<tr>
<td>1</td>
<td>30.0</td>
</tr>
<tr>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Null Values

- The value of an attribute can be unknown (e.g., a rating has not been assigned) or inapplicable (e.g., no spouse).
  - SQL provides a special value null for such situations.
- The presence of null complicates many issues. E.g.:
  - Special operators needed to check if value is/is not null (IS NULL, IS NOT NULL in SQL).
  - Is rating > 8 true or false when rating is equal to null? What about AND, OR and NOT connectives? 3-valued logic (true, false and unknown).
  - Meaning of constructs must be defined carefully. (e.g., WHERE clause eliminates rows that don’t evaluate to true.)
### Outer Join

- A variant of the join that relies on null values:
  
  ```sql
  SELECT c.sid, a.slope-id
  FROM Customers c
  NATURAL LEFT OUTER JOIN Activities a
  ```

- Tuples of Customers table that do not match some tuple in Activities table would normally be excluded from the result; the “left” outer join preserves them with null values for the missing slope-id attribute.

- RIGHT OUTER JOIN
- FULL OUTER JOIN

<table>
<thead>
<tr>
<th>sid</th>
<th>slope-id</th>
</tr>
</thead>
<tbody>
<tr>
<td>2336</td>
<td>s3</td>
</tr>
<tr>
<td>2336</td>
<td>s1</td>
</tr>
<tr>
<td>1887</td>
<td>s2</td>
</tr>
<tr>
<td>1887</td>
<td>s1</td>
</tr>
<tr>
<td>2334</td>
<td>s2</td>
</tr>
<tr>
<td>2339</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### Summary

- Note that aggregation, GROUP BY cannot be expressed in relational algebra
- SQL is relationally complete: all of the operators of the relational algebra can be simulated in SQL.
- Additional features: string comparisons, set membership, arithmetic and grouping, nested queries.