Today's Lecture

• A little bit on views
• Basic SQL queries
• Relational Algebra

Recommended Readings

• Chapter 3
  – Section 3.6.1, 3.7
• Chapter 5
  – Section 5.2.1
• Chapter 4
  – Section 4.1, 4.2
• SQL for Web nerds – Simple queries
  – The link is on the class web page
**Views**

- Physical schema
- Logical/Conceptual schema
- External schema
  - What applications can see
- Views mechanism supports logical data independence

---

**Diagram: VIEWS**

- External
  - YoungActiveStudents, Faculty-info
- Logical
  - Students, Courses, Enrollments, Faculties
- Physical
  - Disk

---

**Views**

- External schema is created by defining views over the logical schema
- Of course, sometimes the logical schema serves well as the external schema but sometimes we need to define new “relations” over existing ones
- A view is a relation, defined in terms of other relations, but its tuples are not stored.
- Virtual views versus Materialized views
Views

CREATE VIEW YoungActiveStudents (name, grade) AS
SELECT S.name, E.grade
FROM Students S, Enrolled E
WHERE S.sid = E.sid and S.age<21

• Views can be dropped using the DROP VIEW command. Similarly, tables can be dropped with DROP TABLE command
• What happens if DROP TABLE is performed and there is a view defined on the table?
  • DROP TABLE command has options to let the user specify this.

Views and Security

• Views can be used to present only the desired information, while hiding details in underlying relation(s).
  – Given YoungActiveStudents, but not Students or Enrolled, we can find young students who are enrolled, but not the cid’s of the courses they are enrolled in.
• Define views and allow only a group of users to access to it
**Relational Model: Summary**

- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
  - Two important ICs: primary and foreign keys
  - In addition, we always have domain constraints.
- Rules to translate ER to relational model
- How do we query relational data?

---

**Querying with SQL**

- SQL is the most popular querying language for relational DBMS
- Basic form:

  ```sql
  SELECT [DISTINCT] c_1, c_2, ..., c_m
  FROM R_1, R_2, ..., R_n
  [WHERE condition]
  ```

- What is the semantics (or meaning) of such a query?
### Example 1 - a simply query

SELECT age
FROM Students
• For every tuple in Students, emit only the age component.

<table>
<thead>
<tr>
<th>Sid</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1124</td>
<td>John</td>
<td>21</td>
</tr>
<tr>
<td>2187</td>
<td>Ann</td>
<td>20</td>
</tr>
<tr>
<td>8992</td>
<td>John</td>
<td>21</td>
</tr>
<tr>
<td>2346</td>
<td>Jane</td>
<td>19</td>
</tr>
</tbody>
</table>

#### Equivalent to:

SELECT age
FROM Students
WHERE TRUE

Note the multiset semantics

### Example 2 - DISTINCT

SELECT DISTINCT age
FROM Students
• For every tuple in Students, emit only the age component. Take the set of resulting values (i.e., remove duplicates)

<table>
<thead>
<tr>
<th>Sid</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1124</td>
<td>John</td>
<td>21</td>
</tr>
<tr>
<td>2187</td>
<td>Ann</td>
<td>20</td>
</tr>
<tr>
<td>8992</td>
<td>John</td>
<td>21</td>
</tr>
<tr>
<td>2346</td>
<td>Jane</td>
<td>19</td>
</tr>
</tbody>
</table>
Example 3 - CONDITION

SELECT DISTINCT name
FROM Students
WHERE Age ≥ 20

- For every tuple in Students, if the value of the age value of that tuple is more than 19, emit the name component of that tuple. Remove duplicates.

<table>
<thead>
<tr>
<th>Sid</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1124</td>
<td>John</td>
<td>21</td>
</tr>
<tr>
<td>2187</td>
<td>Ann</td>
<td>20</td>
</tr>
<tr>
<td>8992</td>
<td>John</td>
<td>21</td>
</tr>
<tr>
<td>2346</td>
<td>Jane</td>
<td>19</td>
</tr>
</tbody>
</table>

Example 4 - Tuple variables

SELECT DISTINCT *
FROM Students S
WHERE S.Age ≥ 20

- The symbol "*" is short for all columns
- The variable S binds to a tuple in Students
- For every tuple S in Students, if the value of the age column of S is more than 20, emit S. Remove duplicates.

<table>
<thead>
<tr>
<th>Sid</th>
<th>Name</th>
<th>Age</th>
<th>S.Sid</th>
<th>S.Name</th>
<th>S.Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1124</td>
<td>John</td>
<td>21</td>
<td>1124</td>
<td>John</td>
<td>21</td>
</tr>
<tr>
<td>2187</td>
<td>Ann</td>
<td>20</td>
<td>2187</td>
<td>Ann</td>
<td>20</td>
</tr>
<tr>
<td>8992</td>
<td>John</td>
<td>21</td>
<td>8992</td>
<td>John</td>
<td>21</td>
</tr>
</tbody>
</table>
Example 5 - Multiple Relations in FROM clause

- What does the following query compute?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade="A"
```

<table>
<thead>
<tr>
<th>Sid</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>53630</td>
<td>Smith</td>
<td>21</td>
</tr>
<tr>
<td>21870</td>
<td>Ann</td>
<td>20</td>
</tr>
<tr>
<td>53831</td>
<td>John</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53831</td>
<td>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53831</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

Condition

- The WHERE clause consists of a boolean combination of conditions using logical connectives AND, OR, NOT.
- Each condition is of the form

  \[
  \text{expression op expression}
  \]

  - \text{expression} is a column name, a constant, or an arithmetic or string expression
  - \text{op} is a comparison operator (\text{<, \leq, =, >, \geq, >})
  - Examples:
    - age > shoesize,
    - name <> “Jane” AND salary*0.1 > 100
**Meaning of an SQL query**

```sql
SELECT [DISTINCT] c_1, c_2, ..., c_m
FROM R_1, R_2, ..., R_n
[WHERE condition]
```

- For every $n$-ary tuple $t$ (one from $R_1$, one from $R_2$, ..., one from $R_n$),
  - if $t$ satisfies condition (i.e., condition evaluates to true), then emit the $c_1, c_2, ..., c_m$ components of $t$.
- Let $Result$ denote the collection of all emitted results.
- If DISTINCT is stated in the SELECT clause, remove duplicates in $Result$ (i.e., $Result$ is a set of tuples. Otherwise, $Result$ is a bag of tuples).

- Note that the order of emitted results is not important.
- Try out on your own on PostgreSQL!

Winter 2003

---

**More on SQL**

- Many more features in SQL! (look at the size of the manual)
- Before we leap ...
  - what is the foundation/properties of SQL?

- Recall that a data model consists of two parts
  - Formalism for describing data
  - Set of operations used to manipulate data
  - Relational algebra, relational calculus are two other different ways of describing operations in the relational model

- relational algebra and relational calculus are terse and formal.
  - Without the verboseness of SQL and good for reasoning
  - Form the basis of SQL
Relational Algebra

- Queries in relational algebra are composed using basic operations (functions)
  - Selection
  - Projection
  - Set-theoretic operations:
    - Union
    - Intersection
    - Set-difference
    - Cross-product
  - Renaming
  - Joins
  - Division

Compositionality

- Each operator takes as input one or two relations and returns a relation as output
- A complex expression is built from basic ones
- Five basic relational operations
  - Select
  - Project
  - Product
  - Union
  - Difference
- In relational algebra (or relational calculus), we assume set semantics (not bag or multiset semantics unless explicitly specified). Duplicate elimination is always performed
Selection - $\sigma_{\text{condition}}(R)$

- Unary operation
  - Input: $R(A_1, \ldots, A_n)$
  - Output relation has attributes $A_1, \ldots, A_n$
- Note:
  - named field notation: we have assumed that the attributes of input relation are known and "passed along" to the output. Easier to understand
  - Strictly speaking, the attributes are not known and we use position numbers instead to refer to the fields. (both are used in SQL)
  - Meaning: Takes a relation $R$ and extracts only rows from $R$ that satisfy the condition.
- Output is always a set (no duplicates)

Example - Select

- $\sigma_{\text{rating} > 6}$ (Hotels)
- Positional notation: $\sigma_{5 > 6}$ (Hotel)

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>rating</th>
<th>capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windsor</td>
<td>54th ave</td>
<td>6.0</td>
<td>135</td>
</tr>
<tr>
<td>Astoria</td>
<td>5th ave</td>
<td>8.0</td>
<td>231</td>
</tr>
<tr>
<td>BestInn</td>
<td>45th st</td>
<td>6.7</td>
<td>28</td>
</tr>
<tr>
<td>ELogde</td>
<td>35 W st</td>
<td>5.6</td>
<td>45</td>
</tr>
<tr>
<td>ELodge</td>
<td>2nd E st</td>
<td>6.0</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>rating</th>
<th>capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astoria</td>
<td>5th ave</td>
<td>8.0</td>
<td>231</td>
</tr>
<tr>
<td>BestInn</td>
<td>45th st</td>
<td>6.7</td>
<td>28</td>
</tr>
</tbody>
</table>
Example - Select

- $\sigma_{\text{rating} > 6 \ \text{AND} \ \text{capacity} > 50}$ (Hotel)

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>rating</th>
<th>capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windsor</td>
<td>54th ave</td>
<td>6.0</td>
<td>135</td>
</tr>
<tr>
<td>Astoria</td>
<td>5th ave</td>
<td>8.0</td>
<td>231</td>
</tr>
<tr>
<td>BestInn</td>
<td>45th st</td>
<td>6.7</td>
<td>28</td>
</tr>
<tr>
<td>ELogde</td>
<td>39 W st</td>
<td>5.6</td>
<td>45</td>
</tr>
<tr>
<td>ELodge</td>
<td>2nd E st</td>
<td>6.0</td>
<td>40</td>
</tr>
</tbody>
</table>

- Is $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c1 \ \text{AND} \ c2}(R)$? Prove or give a counter-example
  - In class

Projection - $\pi_{\text{attribute list}}(R)$

- Unary operation
  - Input: $R(A_1, \ldots, A_n)$
  - Output relation has attributes according to attribute list
  - Meaning: Emits only the attributes stated in attribute list of every tuple in relation R.
- Eliminate duplicates.
Example - Project

- $\pi_{\text{name}, \text{address}}(\text{Hotels})$

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windsor</td>
<td>54th ave</td>
</tr>
<tr>
<td>Astoria</td>
<td>5th ave</td>
</tr>
<tr>
<td>BestInn</td>
<td>45th st</td>
</tr>
<tr>
<td>ELodge</td>
<td>39 W st</td>
</tr>
<tr>
<td>ELodge</td>
<td>2nd E st</td>
</tr>
</tbody>
</table>

- Suppose name and address form the key of Hotels relation, is the cardinality of the output relation the same as the cardinality of Hotels? Why?

Example - Project

- $\pi_{\text{name}}(\text{Hotel})$

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windsor</td>
</tr>
<tr>
<td>Astoria</td>
</tr>
<tr>
<td>BestInn</td>
</tr>
<tr>
<td>ELodge</td>
</tr>
</tbody>
</table>

- Note that there are no duplicates
Set Union - $R \cup S$

- Binary operation
  - Input: $R$ and $S$ must be union-compatible
    - They have the same set of arity, same number of columns
    - The $i$th column of $R$ has the same type as the $i$th column of $S$ for every column $i$.
    - Note that field names are not used in defining union-compatibility though we can also think that $R$ and $S$ is union-compatible if they having the same type (a set of record type).
  - Output has the same type as $R$
  - Meaning: the output consists of the set of all tuples in $R$ and $S$

Example - Set Union

- $\text{Dell.Desktops} \cup \text{IBM.Desktops}$

<table>
<thead>
<tr>
<th>Dell_desktops</th>
<th>Harddisk</th>
<th>Speed</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>20G</td>
<td>500Mhz</td>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>30G</td>
<td>1.0Ghz</td>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>20G</td>
<td>750Mhz</td>
<td>Linux</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IBM_desktops</th>
<th>Harddisk</th>
<th>Speed</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>30G</td>
<td>1.2Ghz</td>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>20G</td>
<td>500Mhz</td>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

All tuples in $R$ occurs in $R \cup S$
All tuples in $S$ occurs in $R \cup S$
$R \cup S$ contains tuples that either occur in $R$ or $S$ (or both).
Example - Set Union

- Dell_Desktops \cup IBM_Desktops

<table>
<thead>
<tr>
<th>Dell_desktops</th>
<th>IBM_desktops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harddisk</td>
<td>Speed</td>
</tr>
<tr>
<td>20G</td>
<td>500Mhz</td>
</tr>
<tr>
<td>30G</td>
<td>1.0Ghz</td>
</tr>
<tr>
<td>20G</td>
<td>750Mhz</td>
</tr>
<tr>
<td>Harddisk</td>
<td>Speed</td>
</tr>
<tr>
<td>20G</td>
<td>500Mhz</td>
</tr>
<tr>
<td>30G</td>
<td>1.0Ghz</td>
</tr>
<tr>
<td>20G</td>
<td>750Mhz</td>
</tr>
</tbody>
</table>

R \cup S = S \cup R (communtativity)
R \cup (S \cup T) = (R \cup S) \cup T (associativity)

Set Difference - R - S

- Binary operation
  - Input: R and S must be union-compatible
  - Output has the same type as R
  - Meaning: output consists of all tuples in R and not in S
**Example - Difference**

- **Dell_Desktops - IBM_Desktops**

<table>
<thead>
<tr>
<th>Harddisk</th>
<th>Speed</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>20G</td>
<td>500Mhz</td>
<td>Windows</td>
</tr>
<tr>
<td>30G</td>
<td>1.0Ghz</td>
<td>Windows</td>
</tr>
<tr>
<td>20G</td>
<td>750Mhz</td>
<td>Linux</td>
</tr>
</tbody>
</table>

- **IBM_Desktops ± Dell_Desktops**

<table>
<thead>
<tr>
<th>Harddisk</th>
<th>Speed</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>30G</td>
<td>1.0Ghz</td>
<td>Windows</td>
</tr>
<tr>
<td>20G</td>
<td>750Mhz</td>
<td>Linux</td>
</tr>
</tbody>
</table>

Winter 2003

---

**Example - Difference**

- **IBM_Desktops – Dell_Desktops**

<table>
<thead>
<tr>
<th>Harddisk</th>
<th>Speed</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>20G</td>
<td>500Mhz</td>
<td>Windows</td>
</tr>
<tr>
<td>30G</td>
<td>1.0Ghz</td>
<td>Windows</td>
</tr>
<tr>
<td>20G</td>
<td>750Mhz</td>
<td>Linux</td>
</tr>
</tbody>
</table>

- **Dell_Desktops – IBM_Desktops**

<table>
<thead>
<tr>
<th>Harddisk</th>
<th>Speed</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>30G</td>
<td>1.0Ghz</td>
<td>Windows</td>
</tr>
<tr>
<td>20G</td>
<td>750Mhz</td>
<td>Linux</td>
</tr>
</tbody>
</table>

Winter 2003

Is it commutative? NO. We have just seen a counter-example.

Is it associative?
Join or Cross Product - $R \times S$

- Join is a generic term for a variety of operations that connect two relations that may not be union-compatible.
  - basic operation is the product, $R \times S$
- Binary operation
  - Input: $R(A_1, ..., A_m), S(B_1, ..., B_n)$
  - Output is a relation with columns $A_1, ..., A_m, B_1, ..., B_n$
    - What happens if $R$ and $S$ contain common attributes?
    - Meaning: concatenates every tuple in $R$ with every tuple in $S$.

Example - Cross Product

<table>
<thead>
<tr>
<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>$c_1$</td>
<td>$d_1$</td>
<td>$e_1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_1$</td>
<td>$e_1$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$d_3$</td>
<td>$e_3$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_2$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_3$</td>
<td>$e_3$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>$e_1$</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>$d_3$</td>
<td>$e_3$</td>
</tr>
</tbody>
</table>

Is it commutative?
Is it associative?
Is it distributive? I.e., $R \times (S \cup T) = (R \times S) \cup (R \times T)$
Example - Cross Product

- What happens if R and S contain common attributes? e.g., R(A,B,C) and S(A,E)
  - Answer varies. We may assume positional suffixes exists to distinguish among conflicts

<table>
<thead>
<tr>
<th>A.1</th>
<th>B</th>
<th>C</th>
<th>A.2</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1</td>
<td>b_1</td>
<td>c_1</td>
<td>d_1</td>
<td>e_1</td>
</tr>
<tr>
<td>a_1</td>
<td>b_1</td>
<td>c_1</td>
<td>d_2</td>
<td>e_2</td>
</tr>
<tr>
<td>a_1</td>
<td>b_1</td>
<td>c_1</td>
<td>d_3</td>
<td>e_3</td>
</tr>
<tr>
<td>a_2</td>
<td>b_2</td>
<td>c_2</td>
<td>d_1</td>
<td>e_1</td>
</tr>
<tr>
<td>a_2</td>
<td>b_2</td>
<td>c_2</td>
<td>d_2</td>
<td>e_2</td>
</tr>
<tr>
<td>a_2</td>
<td>b_2</td>
<td>c_2</td>
<td>d_3</td>
<td>e_3</td>
</tr>
</tbody>
</table>

Winter 2003

More complex queries

- Relational queries can be composed to form more complex queries
- Enrollments(sid, cid, score), Courses(cid, cname, instructor-name)

  - \( \sigma_{\text{score}>80} ( \pi_{\text{sid, grade, instructor-name}} ( \sigma_{\text{Enrollments.cid = Courses.cid}} (\text{Enrollments x Courses}))) \)

- We can now find out the student and instructor pairs where the student scored well (more than 80 pts) in a course taught by the instructor
More complex queries

• Query tree or Operator tree

![Query tree diagram]

Tells us exactly the procedure to take in order to arrive at the answer. Also known as execution plan.

More complex queries

• Find the student and course names where the student scored well (more than 80 pts) in the course

![Query tree diagram]
An alternative plan

- Find the student and course names where the student scored well (more than 80 pts) in the course

\[
\pi_{\text{name, cname}} \\
\sigma_{\text{Enrollments.sid = Students.sid AND Enrollments.cid = Courses.cid AND grade>80}} \\
\]

Which is a better execution plan?

Renaming - \( \rho_X(E) \)

- Renaming is used to give a name to the result of a relational algebra expression.

- Suppose the schema is \( R(A, B, C) \), we write \( \rho_{B \rightarrow D}(R) \) to change the schema to \( R(A, D, C) \)

- Suppose \( R(A,B,C) \) and \( S(D,A) \) in the previous example. We have naming conflict on attribute \( A \) in \( R \times S \). We can write \( \rho_{1 \rightarrow G}(R \times S) \) to correct the ambiguity. Therefore the result has schema \( R \times S(G,B,C,D,A) \)