CSE 444: Database Internals
Lecture 2
Review of the
Relational Model and SQL

Magda Balazinska - CSE 444, Spring 2013
1

Relation Definition

- Database is collection of relations
- Relation R is subset of $S_1 \times S_2 \times \ldots \times S_n$
  - Where $S_i$ is the domain of attribute $i$
  - $n$ is number of attributes of the relation
- Relation is basically a table with rows & columns
  - SQL uses word table to refer to relations

Properties of a Relation

- Each row represents an $n$-tuple of $R$
- Ordering of rows is immaterial (a relation is a set)
- All rows are distinct
- Ordering of columns is significant
  - Because two columns can have same domain
  - But columns are labeled so
  - Applications need not worry about order
  - They can simply use the names
- Domain of each column is a primitive type
- Relation consists of a relation schema and instance

Magda Balazinska - CSE 444, Spring 2013
3

More Definitions

- Relation schema: describes column heads
  - Relation name
  - Name of each field (or column, or attribute)
  - Domain of each field
- Degree (or arity) of relation: nb attributes
- Database schema: set of all relation schemas

Magda Balazinska - CSE 444, Spring 2013
4

Even More Definitions

- Relation instance: concrete table content
  - Set of tuples (also called records) matching the schema
- Cardinality of relation instance: nb tuples
- Database instance: set of all relation instances

Magda Balazinska - CSE 444, Spring 2013
5

Example

- Relation schema
  Supplier(sno: integer, sname: string, scity: string, sstate: string)

- Relation instance

<table>
<thead>
<tr>
<th>sno</th>
<th>sname</th>
<th>scity</th>
<th>sstate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s1</td>
<td>city1</td>
<td>WA</td>
</tr>
<tr>
<td>2</td>
<td>s2</td>
<td>city1</td>
<td>WA</td>
</tr>
<tr>
<td>3</td>
<td>s3</td>
<td>city2</td>
<td>MA</td>
</tr>
<tr>
<td>4</td>
<td>s4</td>
<td>city2</td>
<td>MA</td>
</tr>
</tbody>
</table>

Magda Balazinska - CSE 444, Spring 2013
6
Integrity Constraints

- **Integrity constraint**
  - Condition specified on a database schema
  - Restricts data that can be stored in db instance

- DBMS enforces integrity constraints
  - Ensures only legal database instances exist

- Simplest form of constraint is domain constraint
  - Attribute values must come from attribute domain

Key Constraints

- **Key constraint**: “certain minimal subset of fields is a unique identifier for a tuple”

- **Candidate key**
  - Minimal set of fields
  - That uniquely identify each tuple in a relation

- **Primary key**
  - One candidate key can be selected as primary key

Foreign Key Constraints

- A relation can refer to a tuple in another relation

- **Foreign key**
  - Field that refers to tuples in another relation
  - Typically, this field refers to the primary key of other relation
  - Can pick another field as well

Key Constraint SQL Examples

```sql
CREATE TABLE Part (  
pno integer,  
pname varchar(20),  
psize integer,  
pcolor varchar(20),  
PRIMARY KEY (pno)
);
```

```sql
CREATE TABLE Supply (  
sno integer,  
pno integer,  
qty integer,  
price integer
);
```

```sql
CREATE TABLE Supply (  
sno integer,  
pno integer,  
qty integer,  
price integer,  
PRIMARY KEY (sno,pno)
);
```
Key Constraint SQL Examples

CREATE TABLE Supply(
  sno integer,
  pno integer,
  qty integer,
  price integer,
  PRIMARY KEY (sno, pno),
  FOREIGN KEY (sno) REFERENCES Supplier,
  FOREIGN KEY (pno) REFERENCES Part
);

Key Constraint SQL Examples

CREATE TABLE Supply(
  sno integer,
  pno integer,
  qty integer,
  price integer,
  PRIMARY KEY (sno, pno),
  FOREIGN KEY (sno) REFERENCES Supplier
  ON DELETE NO ACTION,
  FOREIGN KEY (pno) REFERENCES Part
  ON DELETE CASCADE
);

General Constraints

- Table constraints serve to express complex constraints over a single table

CREATE TABLE Part {
  pno integer,
  pname varchar(20),
  psize integer,
  pcolor varchar(20),
  PRIMARY KEY (pno),
  CHECK ( psize > 0 )
};

Note: Also possible to create constraints over many tables

Relational Queries

- Query inputs and outputs are relations

- Query evaluation
  - Input: instances of input relations
  - Output: instance of output relation

Relational Operators

- Selection: $\sigma_{\text{condition}}(S)$
  - Condition is Boolean combination ($\land$, $\lor$) of terms
  - Term is: attr. op constant, attr. op attr.
  - Op is: $<$, $\leq$, $=$, $\neq$, $\geq$, or $>$

- Projection: $\pi_{\text{list-of-attributes}}(S)$

- Union ($\cup$), Intersection ($\cap$), Set difference ($-$), Cross-product or cartesian product ($\times$)

- Join: $R \bowtie \theta S = \sigma_{\theta}(R \times S)$

- Division: $R/S$, Rename $\rho_{\text{F},E}(R(F,E)$

Relational Algebra

- Query language associated with relational model

  - Queries specified in an operational manner
    - A query gives a step-by-step procedure

  - Relational operators
    - Take one or two relation instances as argument
    - Return one relation instance as result
    - Easy to compose into relational algebra expressions
Selection & Projection Examples

<table>
<thead>
<tr>
<th>Patient</th>
<th>(\pi_{\text{zip,disease}}(\text{Patient}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>p1</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
</tr>
<tr>
<td>3</td>
<td>p3</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
</tr>
<tr>
<td>(\sigma_{\text{disease}='\text{heart}'}(\text{Patient}))</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>p2</td>
</tr>
<tr>
<td>2</td>
<td>p4</td>
</tr>
</tbody>
</table>

Relational Operators

- **Selection**: \(\sigma_{\text{condition}(S)}\)
  - Condition is Boolean combination (\&, \lor) of terms
  - Term is: attr. op constant, attr. op attr.
  - Op is: <, <=, =, \# =, >, or >
- **Projection**: \(\pi_{\text{list-of-attributes}}(S)\)
- Union (\(\cup\)), Intersection (\(\cap\)), Set difference (\(-\))
- Cross-product or cartesian product (\(\times\))
- Join: \(R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)\)
- Division: \(R/S\)
- Rename \(\rho(R(F),E)\)

Cross-Product Example

<table>
<thead>
<tr>
<th>AnonPatient</th>
<th>Voters</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>zip</td>
</tr>
<tr>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

\(P \times V\)

Theta-Join Example

<table>
<thead>
<tr>
<th>AnonPatient</th>
<th>Voters</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>zip</td>
</tr>
<tr>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

\(P \bowtie_{P.\text{age}=V.\text{age} \land P.\text{zip}=V.\text{zip}}\)

Equijoin Example

<table>
<thead>
<tr>
<th>AnonPatient</th>
<th>Voters</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>zip</td>
</tr>
<tr>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

\(P \bowtie_{P.\text{age}=V.\text{age}}\)
Natural Join Example

AnonPatient P

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
</tbody>
</table>

Voters V

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

P ⋈ V

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>p1</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
</tr>
</tbody>
</table>

More Joins

- **Outer join**
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes

- **Variants**
  - Left outer join
  - Right outer join
  - Full outer join

Example of Algebra Queries

Q1: Names of patients who have heart disease
\[ \pi_{\text{name}}(\text{Voter} \bowtie \sigma_{\text{disease}='heart'}(\text{AnonPatient})) \]

More Examples

Relations

- Supplier(sno, sname, scity, sstate)
- Part(pno, pname, psize, pcolor)
- Supply(sno, pno, qty, price)

Q2: Name of supplier of parts with size greater than 10
\[ \pi_{\text{sname}}(\text{Supplier} \bowtie \text{Supply} \bowtie \sigma_{\text{psize}>10}(\text{Part})) \]

Q3: Name of supplier of red parts or parts with size greater than 10
\[ \pi_{\text{sname}}(\text{Supplier} \bowtie \text{Supply} \bowtie \sigma_{\text{psize}>10}(\text{Part}) \cup \sigma_{\text{pcolor}='red'}(\text{Part})) \]

(Many more examples in the book)

Logical Query Plans

An RA expression but represented as a tree

```
π_{\text{name}}(σ_{\text{psize}>10}(\text{Supply} \bowtie \text{Part}))
```

(Many more examples in the book)
Extended Operators of Relational Algebra

- Duplicate elimination (δ)
  - Since commercial DBMSs operate on multisets not sets
- Aggregate operators (∑)
  - Min, max, sum, average, count
- Grouping operators (∑)
  - Partitions tuples of a relation into "groups"
  - Aggregates can then be applied to groups
- Sort operator (∑)

Structured Query Language: SQL

- Influenced by relational calculus (see 344)
- Declarative query language
- Multiple aspects of the language
  - Data definition language
    - Statements to create, modify tables and views
  - Data manipulation language
    - Statements to issue queries, insert, delete data
    - More

SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT <attributes>
FROM <one or more relations>
WHERE <conditions>
```

Simple SQL Query

```
SELECT PName, Price, Manufacturer
FROM Product
WHERE Price > 100
```

Details

- Case insensitive:
  - Same: SELECT Select select
  - Same: Product product
  - Different: 'Seattle'  'seattle'
- Constants:
  - 'abc' - yes
  - "abc" - no
Eliminating Duplicates

```
SELECT DISTINCT category
FROM Product
```

Compare to:

```
SELECT category
FROM Product
```

Ordering the Results

```
SELECT pname, price, manufacturer
FROM Product
WHERE category='gizmo' AND price > 50
ORDER BY price, pname
```

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering is ascending, unless you specify the DESC keyword.

Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer=CName AND Country='Japan'
AND Price <= 200
```

Tuple Variables

```
SELECT DISTINCT pname, address
FROM Person, Company
WHERE worksfor = cname
```

Which address?

```
SELECT Person.pname, Company.address
FROM Person, Company
WHERE Person.worksfor = Company.cname
```

```
SELECT DISTINCT x.pname, y.address
FROM Person AS x, Company AS y
WHERE x.worksfor = y.cname
```

Nested Queries

• Nested query
  – Query that has another query embedded within it
  – The embedded query is called a subquery

• Why do we need them?
  – Enables to refer to a table that must itself be computed

• Subqueries can appear in
  – WHERE clause (common)
  – FROM clause (less common)
  – HAVING clause (less common)

Subqueries Returning Relations

Return cities where one can find companies that manufacture products bought by Joe Blow

```
SELECT Company.city
FROM Company
WHERE Company.name IN
  (SELECT Product.maker
   FROM Purchase, Product
   WHERE Product.pname=Purchase.product
   AND Purchase.buyer = 'Joe Blow')
```
Subqueries Returning Relations

You can also use:  
- \( s > \text{ALL } R \)
- \( s > \text{ANY } R \)
- \( \exists \text{R} \)

Find products that are more expensive than all those produced by “Gizmo-Works”:

```sql
SELECT name
FROM Product
WHERE price > (SELECT price
  FROM Purchase
WHERE maker='Gizmo-Works')
```

Correlated Queries

Find movies whose title appears more than once:

```sql
SELECT DISTINCT title
FROM Movie AS x
WHERE year <> (SELECT year
  FROM Movie
WHERE title = x.title)
```

Aggregation

SQL supports several aggregation operations:
- sum, count, min, max, avg

Except count, all aggregations apply to a single attribute.

```sql
SELECT avg(price)
FROM Product
WHERE maker='Toyota'
```

Grouping and Aggregation

Conceptual evaluation steps:
1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes \( a_1, \ldots, a_k \)
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Read more about it in the book...

```sql
SELECT S
FROM R_1, \ldots, R_n
WHERE C1
GROUP BY a_1, \ldots, a_k
HAVING C2
```

From SQL to RA

Product(pid, name, price)  
Purchase(pid, cid, store)  
Customer(cid, name, city)

```sql
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
  x.price > 100 and z.city = 'Seattle'
```
From SQL to RA

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

\[ \delta \]
\[ \sigma \text{price}=100 \text{ and city='Seattle'} \]
\[ \pi \text{x.name, z.name} \]
\[ \delta \text{cid=cid} \]

Customer

\[ \prod \text{Product(pid, name, price)} \]
\[ \Pi \text{Purchase(pid, cid, store)} \]
\[ \sigma \text{city='Seattle'} \]

An Equivalent Expression

Query optimization = finding cheaper, equivalent expressions

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

\[ \delta \]
\[ \sigma \text{price}>100 \]
\[ \pi \text{x.name, z.name} \]
\[ \delta \text{cid=cid} \]

Customer

Extended RA: Operators on Bags

• Duplicate elimination \( \delta \)
• Grouping \( \gamma \)
• Sorting \( \tau \)

Logical Query Plan

SELECT city, count(*)
FROM sales
GROUP BY city
HAVING \( \sum \text{price} > 100 \)

Typical Plan for Block (1/2)

Typical Plan For Block (2/2)
Benefits of Relational Model

- Relational model facilitates **physical data independence**
  - Can change how data is organized on disk without affecting applications

- Relational model facilitates a high level of **logical data independence**
  - Can change the logical schema without affecting applications (not 100%, consider updates)