Database Management Systems
CS 457/657

Lectures 7: Relational Algebra (cont’d), ER Diagram
Where We Are

• Motivation for using a DBMS for managing data
• SQL, SQL, SQL
  – Declaring the schema for our data (CREATE TABLE)
  – Inserting data one row at a time or in bulk (INSERT/.import)
  – Modifying the schema and updating the data (ALTER/UPDATE)
  – Querying the data (SELECT)

• Next step: More knowledge of how DBMSs work
  – Client-server architecture
  – Relational algebra and query execution
Query Evaluation Steps

1. **Parse & Check Query**
   - Translate query string into internal representation
   - Check syntax, access control, table names, etc.

2. **Decide how best to answer query: query optimization**
   - Logical plan → physical plan

3. **Query Execution**

4. **Return Results**

SQL query
Set Difference

- $A = \{1, 2, 3\}, \ B = \{2, 4\}$
- $A \text{ and } B = \{2\}$
- $A - B = \{1, 3\}$
- $A - (A - B) = \{1, 2, 3\} - \{1, 3\} = \{2\} = A \text{ and } B$
- **WRONG:** $A - (A - B) \neq A - A + B = B \ldots$
Logical Query Plan

```
SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100

T1, T2, T3 = temporary tables
```

Diagram:

```
T1(city, p, c)
\[\gamma_{\text{city, sum(price)}}(\rightarrow p, \text{count(*)} \rightarrow c)\]
\[\sigma_{p > 100}(\rightarrow \Pi_{\text{city, c}})\]
\[\text{sales(product, city, price)}\]
```
Typical Plan for Block (1/2)

\[ \ldots \]

\[ \pi \text{ fields} \]

\[ \sigma \text{ selection condition} \]

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ R \quad S \]

\{ SELECT-PROJECT-JOIN Query \}
Typical Plan For Block (2/2)

\[ \text{having}_{\text{condition}} \]
\[ \gamma \text{ fields, sum/count/min/max(fields)} \]
\[ \pi \text{ fields} \]
\[ \sigma \text{ selection condition} \]
\[ \text{join condition} \]
\[ \ldots \]
\[ \ldots \]
How about Subqueries?

```sql
SELECT  Q.sno
FROM    Supplier Q
WHERE   Q.sstate = 'WA'
        and not exists
            (SELECT *
             FROM  Supply P
             WHERE  P.sno = Q.sno
                    and P.price > 100)
```
How about Subqueries?

```
SELECT  Q.sno
FROM    Supplier Q
WHERE   Q.sstate = 'WA'
        and not exists
        (SELECT *
         FROM  Supply P
         WHERE P.sno = Q.sno
         and P.price > 100)
```
How about Subqueries?

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
     and not exists
     (SELECT *
         FROM Supply P
         WHERE P.sno = Q.sno
             and P.price > 100)
```

De-Correlation

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
     and Q.sno not in
     (SELECT P.sno
         FROM Supply P
         WHERE P.price > 100)
```
How about Subqueries?

(SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA')
EXCEPT
(SELECT P.sno
FROM Supply P
WHERE P.price > 100)
EXCEPT = set difference

SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
and Q.sno not in
(SELECT P.sno
FROM Supply P
WHERE P.price > 100)
How about Subqueries?

```
(SELECT Q.sno
 FROM Supplier Q
 WHERE Q.sstate = 'WA')
 EXCEPT
(SELECT P.sno
 FROM Supply P
 WHERE P.price > 100)
```

Finally...

- \( \pi_{sno} \)
- \( \sigma_{sstate=\text{'WA'}} \)
- \( \sigma_{\text{Price} > 100} \)

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
From Logical Plans to Physical Plans
Query Evaluation Steps Review

- Parse & Rewrite Query
- Select Logical Plan
- Select Physical Plan
- Query Execution

Query optimization

SQL query → Logical plan → Physical plan

Disk
Example

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'
```

Give a relational algebra expression for this query
Relational Algebra

\[ \pi_{\text{sname}}(\sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA} \land \text{pno}=2}(\text{Supplier} \bowtie_{\text{sid} = \text{sid}} \text{Supply})) \]
Relational Algebra

Relational algebra expression is also called the “logical query plan”

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
A physical query plan is a logical query plan annotated with physical implementation details.

Physical Query Plan 1

\[
\begin{align*}
\sigma_{\text{scity}= 'Seattle' \land \text{sstate}= 'WA' \land \text{pno}=2} \quad & \quad \pi_{\text{sname}} \\
\text{SELECT } \text{sname} \\
\text{FROM Supplier } x, \text{ Supply } y \\
\text{WHERE } x.\text{sid} = y.\text{sid} \\
& \quad \text{and } y.\text{pno} = 2 \\
& \quad \text{and } x.\text{scity} = 'Seattle' \\
& \quad \text{and } x.\text{sstate} = 'WA'
\end{align*}
\]
**Physical Query Plan 2**

Supplier\((sid, sname, scity, sstate)\)
Supply\((sid, pno, quantity)\)

\[
\begin{align*}
\text{(On the fly)} & \quad \pi_{\text{sname}} \\
\text{(On the fly)} & \quad \sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA} \land pno=2} \\
\text{(Hash join)} & \quad \text{sid} = \text{sid}
\end{align*}
\]

\[
\text{SELECT } \text{sname} \\
\text{FROM Supplier } x, \text{ Supply } y \\
\text{WHERE } x.\text{sid} = y.\text{sid} \\
\text{and } y.\text{pno} = 2 \\
\text{and } x.\text{scity} = \text{Seattle'} \\
\text{and } x.\text{sstate} = \text{WA'}
\]
Physical Query Plan 3

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Different but equivalent logical query plan; different physical plan

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'

(On the fly)

(Scan & write to T1)

(Scan & write to T2)

Supplier (File scan)

Supply (File scan)
Query Optimization Problem

• For each SQL query… many logical plans

• For each logical plan… many physical plans

• How do find a fast physical plan?
  – Will discuss in a few lectures
  – A lot more details, low-level system stuff
Query Execution
Pipelined Execution

• Tuples generated by an operator are immediately sent to the parent

• Benefits:
  – No operator synchronization issues
  – No need to buffer tuples between operators
  – Saves cost of writing intermediate data to disk
  – Saves cost of reading intermediate data from disk

• This approach is used whenever possible
Intermediate Tuple Materialization

- Tuples generated by an operator are written to disk in an intermediate table
- No direct benefit
- Necessary:
  - For certain operator implementations
  - When we don’t have enough memory
Intermediate Tuple Materialization

(On the fly)

(Sort-merge join)

(Scan: write to T1)

(Scan: write to T2)

\[ \pi_{\text{sname}} \]

\[ \sigma_{\text{sscity}=\text{Seattle} \land \text{sstate}=\text{WA}} \]

\[ \sigma_{\text{pno}=2} \]

Suppliers (File scan)

Supplies (File scan)
Query Execution Bottom Line

- SQL query transformed into **physical plan**
  - **Access path selection** for each relation
    - Scan the relation or use an index (discussed later)
  - **Implementation choice** for each operator
    - Nested loop join, hash join, etc.
  - **Scheduling decisions** for operators
    - Pipelined execution or intermediate materialization
Where are we?

• We are very close to the internal details of a DBMS
  – System architecture, Storage management, Memory management

• Before digging into them, let’s spend some time on understanding how a DBMS user designs her databases
  – We already know users will use SQL to implement databases
Database Management Systems
CS 457/657

Lectures 7: ER Diagram
Database Design

• Why do we need it?
  – Need a way to model real world entities in terms of relations
  – Not easy to go from real-world entities to a database schema

• Consider issues such as:
  – What entities to model
  – How entities are related
  – What constraints exist in the domain
  – How to achieve good designs

• Several formalisms exists
  – We discuss E/R diagrams
**Database Design Process**

- **Conceptual Model:**
  - **Product**
  - **Makes**
  - **Company**
  - **Name**
  - **Price**
  - **Name**
  - **Address**

- **Relational Model:**
  - Tables + constraints
  - And also functional dep.

- **Normalization:**
  - Eliminates anomalies

- **Conceptual Schema**

- **Physical storage details**
  - **Physical Schema**
Entity / Relationship Diagrams

- Entity set = a class
  - An entity = an object

- Attribute

- Relationship

Diagram:
- Product
- City
- Makes
Keys in E/R Diagrams

- Every entity set must have a key

```
Product
  • name
  • price
```
What is a Relation?

- A mathematical definition:
  - if $A$, $B$ are sets, then a relation $R$ is a subset of $A \times B$

- $A=\{1,2,3\}$, $B=\{a,b,c,d\}$,
  - $A \times B = \{(1,a),(1,b), \ldots, (3,d)\}$
  - $R = \{(1,a), (1,c), (3,b)\}$

- **makes** is a subset of **Product × Company**:

![Diagram showing relation between Product, makes, and Company]
Multiplicity of E/R Relations

- **one-one:**
  - (name, id)

- **many-one**
  - (employee, company)

- **many-many**
  - (book, author)
What does this say?
Multi-way Relationships

• How do we model a purchase relationship between buyers, products and stores?

  - Product
  - Purchase
  - Person
  - Store

• Can still model as a mathematical set (Q. how ?)

• A. As a set of triples $\subseteq \text{Person } \times \text{Product } \times \text{Store}$
Q: What does the arrow mean?

A: A given person buys a given product from at most one store

[Arrow pointing to E means that if we select one entity from each of the other entity sets in the relationship, those entities are related to at most one entity in E]
Q: What does the arrow mean?

A: A given person buys a given product from at most one store AND every store sells to every person at most one product.
Converting Multi-way Relationships to Binary

- **Purchase**
- **ProductOf**
- **StoreOf**
- **BuyerOf**
- **date**
- **Product**
- **Store**
- **Person**

Arrows go in which direction?
Converting Multi-way Relationships to Binary

- date
- Purchase
- StoreOf
- BuyerOf
- ProductOf
- Product
- Store
- Person

Make sure you understand why!