ICS 624 Spring 2013

Overview of Database Systems

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
Outline

• The Data Management Problem
• Data, Databases and DBMSs
• The Relational Model
• Transactions
• Structured Query Language
• Relational Algebra
• Query Processing in DBMSs
The Data Management Problem

Where is the photo I took last Christmas?
Where did I read about “Turing Machines”?
Where is the invoice for this computer?
Which product is the most profitable?

User

Queries

Data

User Queries Data
What is ``data’’?

- **Data** are known facts that can be recorded and that have implicit meaning.

- Three broad categories of data
  - Structured data
  - Semi-structured data
  - Unstructured data

- ``Structure’’ of data refers to the organization within the data that is identifiable.
What is a database?

- A **database**: a collection of related data.
  - Represents some aspect of the real world (aka universe of discourse).
  - Logically coherent collection of data
  - Designed and built for specific purpose
- A **data model** is a collection of concepts for describing/organizing the data.
- A **schema** is a description of a particular collection of data, using the a given data model.
The Relational Data Model

• *Relational database*: a set of *relations*

• A *relation* is made up of 2 parts:
  – *Instance*: a *table*, with rows and columns.
    
    #Rows = *cardinality*, #fields = *degree* / *arity*.
  – *Schema*: specifies name of relation, plus name and *domain/type* of each column or attribute.
    
    • E.G. Students(sid: string, name: string, login: string, age: integer, gpa: real).

• Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).
Example Relations

- **Sailors**
  sid integer, sname string, rating integer, age real

- **Boats**
  bid integer, bname string, color string

- **Reserves**
  sid integer, bid string, day date

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

<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>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>Blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>Red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>Red</td>
</tr>
</tbody>
</table>
Why is the relational model useful?

- Supports simple and powerful query capabilities!
- Structured Query Language (SQL)

```
SELECT S.sname 
FROM   Students S 
WHERE  S.gpa>3.5
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>
What is a DBMS?

• A **database management system (DBMS)** is a **collection of programs** that enables users to
  – Create new DBs and specify the structure using data definition language (DDL)
  – Query data using a query language or data manipulation language (DML)
  – Store very large amounts of data
  – Support **durability** in the face of failures, errors, misuse
  – Control **concurrent** access to data from many users
Types of Databases

- On-line Transaction Processing (OLTP)
  - Banking
  - Airline reservations
  - Corporate records

- On-line Analytical Processing (OLAP)
  - Data warehouses, data marts
  - Business intelligence (BI)

- Specialized databases
  - Multimedia

- XML
- Geographical Information Systems (GIS)
- Real-time databases (telecom industry)

- Special Applications
  - Customer Relationship Management (CRM)
  - Enterprise Resource Planning (ERP)

- Hosted DB Services
  - Amazon, Salesforce
Transactions

• A transaction is the DBMS’s abstract view of a user program: a sequence of reads and writes.
  – Eg. User 1 views available seats and reserves seat 22A.

• A DBMS supports multiple users, ie, multiple transactions may be running concurrently.
  – Eg. User 2 views available seats and reserves seat 22A.
  – Eg. User 3 views available seats and reserves seat 23D.
ACID Properties of Transactions

- **Atomicity**: all-or-nothing execution of transactions
- **Consistency**: constraints on data elements is preserved
- **Isolation**: each transaction executes as if no other transaction is executing concurrently
- **Durability**: effect of an executed transaction must never be lost
A Bit of History

- 1970 Edgar F Codd (aka “Ted”) invented the relational model in the seminal paper “A Relational Model of Data for Large Shared Data Banks”
  - Main concept: relation = a table with rows and columns.
  - Every relation has a schema, which describes the columns.
- Prior 1970, no standard data model.
  - Network model used by Codasyl
  - Hierarchical model used by IMS
- After 1970, IBM built System R as proof-of-concept for relational model and used SQL as the query language. SQL eventually became a standard.
Basic SQL Query

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

- **relation-list** A list of relation names (possibly with a range-variable after each name).
- **target-list** A list of attributes of relations in relation-list.
- **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.
- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are not eliminated!
Example Q1

**SELECT** S.sname

**FROM**  Sailors S, Reserves R

**WHERE**  S.sid=R.sid AND bid=103

Without range variables

**SELECT** sname

**FROM**  Sailors, Reserves

**WHERE**  Sailors.sid=Reserves.sid 
AND bid=103

• Range variables really needed only if the same relation appears twice in the FROM clause.

• Good style to always use range variables
Conceptual Evaluation Strategy

• Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  1. Compute the cross-product of relation-list.
  2. Discard resulting tuples if they fail qualifications.
  3. Delete attributes that are not in target-list.
  4. If DISTINCT is specified, eliminate duplicate rows.

• This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute the same answers.
Example Q1: conceptual evaluation

**Conceptual Evaluation Steps:**
1. Compute cross-product
2. Discard disqualified tuples
3. Delete unwanted attributes
4. If `DISTINCT` is specified, eliminate duplicate rows.

```sql
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND bid=103
```

<table>
<thead>
<tr>
<th>S.sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
<th>R.sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S.sid</th>
<th>surname</th>
<th>rating</th>
<th>age</th>
<th>R.sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

Sname: Rusty
Relational Algebra

• Basic operations:
  – **Selection** \((\sigma)\) Selects a subset of rows from relation.
  – **Projection** \((\pi)\) Deletes unwanted columns from relation.
  – **Cross-product** \((\times)\) Allows us to combine two relations.
  – **Set-difference** \((-)\) Tuples in reln. 1, but not in reln. 2.
  – **Union** \((U)\) Tuples in reln. 1 and in reln. 2.

• Additional operations:
  – Intersection, **join**, division, renaming: Not essential, but (very!) useful.

• Since each operation returns a relation, operations can be composed! (Algebra is “closed”.)
Projection

- Deletes attributes that are not in *projection list*.
- **Schema** of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to **eliminate duplicates**! (Why??)
- Note: real systems typically don’t do duplicate elimination unless the user explicitly asks for it. (Why not?)

<table>
<thead>
<tr>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yuppy</td>
<td>9</td>
</tr>
<tr>
<td>Lubber</td>
<td>8</td>
</tr>
<tr>
<td>Guppy</td>
<td>5</td>
</tr>
<tr>
<td>Rusty</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>35.0</td>
</tr>
<tr>
<td>55.5</td>
</tr>
<tr>
<td>35.0</td>
</tr>
<tr>
<td>35.0</td>
</tr>
</tbody>
</table>
Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)
Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be **union-compatible**:
  - Same number of fields.
  - ‘Corresponding’ fields have the same type.
- What is the **schema** of result?

<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>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>Yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>Guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

S1 U S2

<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>28</td>
<td>Yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>Guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Intersection & Set-Difference

**S1 ∩ S2**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**S1 − S2**

<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>
</tbody>
</table>

**S1**

<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>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

**S2**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>Yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>Guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Cross-Product

• Consider the cross product of $S_1$ with $R_1$
• Each row of $S_1$ is paired with each row of $R_1$.
• *Result schema* has one field per field of $S_1$ and $R_1$, with field names `inherited` if possible.
  – *Conflict*: Both $S_1$ and $R_1$ have a field called *sid*.
  – Rename to sid1 and sid2

<table>
<thead>
<tr>
<th>$S_1$</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
<td>rating</td>
<td>age</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$R_1$</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>bid</td>
<td>day</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S_1 \times R_1$</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sname</td>
<td>rating</td>
<td>age</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Joins

- **Condition Join**: $R \bowtie_c S = \sigma_c (R \times S)$
- **Result schema** same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

\[
S_1 \bowtie \ S_1.\text{sid} < R_1.\text{sid} \ R_1
\]

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Equi-Joins & Natural Joins

• **Equi-join**: A special case of condition join where the condition \( c \) contains only *equalities*.
  
  – **Result schema** similar to cross-product, but only one copy of fields for which equality is specified.

• **Natural Join**: Equi-join on *all* common fields.

\[
S_1 \bowtie_{sid} R_1
\]

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Query

Parse Query

Enumerate Plans

Estimate Cost

Choose Best Plan

Evaluate Query Plan

Result

SELECT * FROM Reserves WHERE sid=101

σ_{Sid=101}

Reserves

SCAN (sid=101)

Reserves

IDXSCAN (sid=101)

Index(sid)

fetch

Evaluate Plan A

Pick B

Optimizer

Evaluate Plan B

Pick B

Reserves

Reserves

32.0

25.0
Parse Query

- Input: SQL
  - Eg. SELECT-FROM-WHERE, CREATE TABLE, DROP TABLE statements
- Output: Some data structure to represent the "query"
  - Relational algebra?
- Also checks syntax, resolves aliases, binds names in SQL to objects in the catalog
- How?
Enumerate Plans

- **Input**: a data structure representing the “query”
- **Output**: a collection of equivalent query evaluation plans
- **Query Execution Plan (QEP)**: tree of database operators.
  - high-level: RA operators are used
  - low-level: RA operators with particular implementation algorithm.
- **Plan enumeration**: find equivalent plans
  - Different QEPs that return the same results
  - Query rewriting: transformation of one QEP to another equivalent QEP.
Estimate Cost

• **Input**: a collection of equivalent query evaluation plans
• **Output**: a cost estimate for each QEP in the collection
• **Cost estimation**: a mapping of a QEP to a cost
  – **Cost Model**: a model of what counts in the cost estimate. Eg. Disk accesses, CPU cost ...
• **Statistics about the data and the hardware are used.**
Choose Best Plan

• **Input**: a collection of equivalent query evaluation plans and their cost estimate

• **Output**: best QEP in the collection

• The steps: enumerate plans, estimate cost, choose best plan collectively called the:

• **Query Optimizer**:
  – Explores the space of equivalent plan for a query
  – Chooses the best plan according to a cost model
Evaluate Query Plan

- **Input**: a QEP (hopefully the best)
- **Output**: Query results
- Often includes a “code generation” step to generate a lower level QEP in executable “code”.
- **Query evaluation engine** is a “virtual machine” that executes some code representing low level QEP.
Query Execution Plans (QEPs)

- A **tree** of database operators: each operator is a RA operator with specific implementation

- **Selection** $\sigma$: Index Scan or Table Scan

- **Projection** $\pi$:
  - Without DISTINCT: Table Scan
  - With DISTINCT: requires sorting or index scan

- **Join** $\Join$:
  - Nested loop joins (naïve)
  - Index nested loop joins
  - Sort merge joins

- **Sort**:
  - In-memory sort
  - External sort
QEP Examples

```sql
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND R.bid=100 AND S.rating>5
```
Access Paths

• **Access path** is a method of retrieving tuples. Eg. Given a query with a selection condition:
  - File or table scan
  - Index scan

• **Index matching problem:** given a selection condition, which indexes can be used for the selection, i.e., matches the selection?
  - Selection condition normalized to conjunctive normal form (CNF), where each term is a *conjunct*
  - Eg. \((\text{day}<8/9/94 \ \text{AND} \ rname='Paul') \ \text{OR} \ \text{bid}=5 \ \text{OR} \ \text{sid}=3\)
  - **CNF:** \((\text{day}<8/9/94 \ \text{OR} \ \text{bid}=5 \ \text{OR} \ \text{sid}=3) \ \text{AND} \ (rname='Paul' \ \text{OR} \ \text{bid}=5 \ \text{OR} \ \text{sid}=3)\)
Index Matching

- A **tree index** matches a selection condition if the selection condition is a prefix of the index search key.
- A **hash index** matches a selection condition if the selection condition has a term `attribute=value` for every attribute in the index search key.

Q1: $\sigma_{a=5 \text{ AND } b=3}$
Q2: $\sigma_{a=5 \text{ AND } b>6}$
Q3: $\sigma_{b=3}$
Q4: $\sigma_{a=5 \text{ AND } b=3 \text{ AND } c=5}$
Q5: $\sigma_{a>5 \text{ AND } b=3 \text{ AND } c=5}$