ICS 624 Spring 2011
Overview of DB & IR

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

### R1
<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>

### 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>

### B1
<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>
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 \( op \) const or Attr1 \( op \) Attr2, where \( op \) 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

\[
\text{SELECT S.sname} \\
\text{FROM Sailors S, Reserves R} \\
\text{WHERE S.sid=R.sid AND bid=103}
\]

Without range variables

\[
\text{SELECT sname} \\
\text{FROM Sailors, Reserves} \\
\text{WHERE Sailors.sid=Reserves.sid AND bid=103}
\]

• 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.

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

<table>
<thead>
<tr>
<th>S.sid</th>
<th>sname</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>sname</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?)
Selection

• Selects rows that satisfy \textit{selection condition}.
• No duplicates in result! (Why?)
• \textit{Schema} of result identical to schema of (only) input relation.
• \textit{Result} relation can be the \textit{input} for another relational algebra operation! (\textit{Operator composition}.)

\begin{tabular}{|c|c|c|c|}
\hline
\text{sid} & \text{sname} & \text{rating} & \text{age} \\
\hline
28 & Yuppy & 9 & 35.0 \\
\hline
31 & Lubber & 8 & 55.5 \\
\hline
44 & Guppy & 5 & 35.0 \\
\hline
58 & Rusty & 10 & 35.0 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|}
\hline
\text{sid} & \text{sname} & \text{rating} & \text{age} \\
\hline
28 & Yuppy & 9 & 35.0 \\
\hline
31 & Lubber & 8 & 55.5 \\
\hline
44 & Guppy & 5 & 35.0 \\
\hline
58 & Rusty & 10 & 35.0 \\
\hline
\end{tabular}
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>

<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>
Intersection & Set-Difference

**S1 \( \cap \) 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 \( \setminus \) 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 S1 with R1
- Each row of S1 is paired with each row of R1.
- **Result schema** has one field per field of S1 and R1, with field names ‘inherited’ if possible.
  - *Conflicted*: Both S1 and R1 have a field called *sid*.
  - Rename to sid1 and sid2

<table>
<thead>
<tr>
<th>S1</th>
<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>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R1</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<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>S1 × R1</th>
<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>22</td>
<td>101</td>
<td>10/10/96</td>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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*.

\[
S_l \bowtie S_l.\text{sid} < R_l.\text{sid} R_l
\]

<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>
SELECT * FROM Reserves WHERE sid=101

\[ \sigma_{\text{Sid}=101} \]

- **Plan A**: SCAN (sid=101)
  - Cost: 32.0

- **Plan B**: IDXSCAN (sid=101)
  - Cost: 25.0

**Optimizer**
- Pick B

**Evaluate Query Plan**
- Evaluate Plan A

**Result**
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**

**SELECT** S.sname  
**FROM** Reserves R, Sailors S  
**WHERE** R.sid=S.sid **AND** R.bid=100 **AND** S.rating>5

\[
\begin{align*}
\pi_{\text{S.sname}} & \quad \sigma_{\text{S.rating}>5 \text{ AND } R.bid=100} \\
\sigma_{\text{S.rating}>5} & \quad \sigma_{\text{R.bid}=100} \\
\text{R.sid=S.sid} & \quad \text{R.sid=S.sid} \\
\text{Reserves} & \quad \text{Sailors} \quad \text{On the fly} \quad \text{Reserves} \quad \text{Sailors} \quad \text{Nested Loop Join} \\
\text{(SCAN)} & \quad \text{(SCAN)} \quad \text{Temp T1}
\end{align*}
\]
Access Paths

- An **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. \((\text{day}<8/9/94 \text{ AND } \text{rname}='Paul') \text{ OR } \text{bid}=5 \text{ OR } \text{sid}=3\)
  - **CNF**: \((\text{day}<8/9/94 \text{ OR } \text{bid}=5 \text{ OR } \text{sid}=3) \text{ AND } (\text{rname}='Paul' \text{ OR } \text{bid}=5 \text{ OR } \text{sid}=3)\)
Index Matching

Q1: \( \sigma \ a=5 \ AND \ b=3 \)
Q2: \( \sigma \ a=5 \ AND \ b>6 \)
Q3: \( \sigma \ b=3 \)
Q4: \( \sigma \ a=5 \ AND \ b=3 \ AND \ c=5 \)
Q5: \( \sigma \ a>5 \ AND \ b=3 \ AND \ c=5 \)

I1: Tree Index (a,b,c)
I2: Tree Index (b,c,d)
I3: Hash Index (a,b,c)

• 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.
Unstructured Text Data

• Field of “Information Retrieval”

• Data Model
  – Collection of documents
  – Each document is a bag of words (aka terms)

• Query Model
  – **Keyword** + Boolean Combinations
  – Eg. DBMS and SQL and tutorial

• Details:
  – Not all words are equal. “**Stop words**” (eg. “the”, “a”, “his” ...) are ignored.
  – **Stemming** : convert words to their basic form. Eg. “Surfing”, “surfled” becomes “surf”
Inverted Indexes

• Recall: an index is a mapping of search key to data entries
  – What is the search key?
  – What is the data entry?

• Inverted Index:
  – For each term store a list of postings
  – A posting consists of <docid, position> pairs

### Posting lists

- **DBMS**
  - `doc01`: 10, 18, 20
  - `doc02`: 5, 38
  - `doc01`: 13

- **SQL**
  - `doc06`: 1, 12
  - `doc09`: 4, 9
  - `doc20`: 12

- **trigger**
  - `doc01`: 12, 15
  - `doc09`: 14, 21, 25
  - `doc10`: 11, 55

What is the data in an inverted index sorted on?
Lookups using Inverted Indexes

- Given a single keyword query “k” (eg. SQL)
  - Find k in the lexicon
  - Retrieve the posting list for k
  - Scan posting list for document IDs [and positions]

- What if the query is “k1 and k2”?
  - Retrieve document IDs for k1 and k2
  - Perform intersection
Too Many Matching Documents

• Rank the results by “relevance”!
• Vector-Space Model
  – **Documents** are **vectors** in high-dimensional space
  – Each dimension in the vector represents a term
  – **Queries** are represented as **vectors** similarly
  – **Vector distance** (dot product) between query vector and document vector gives ranking criteria
  – **Weights** can be used to tweak relevance
• PageRank (later)

Star

Doc about movie stars

Doc about astronomy

Doc about behavior

Diet
How good are the retrieved docs?

- **Precision**: Fraction of retrieved docs that are relevant to user’s information need
- **Recall**: Fraction of relevant docs in collection that are retrieved