An Introduction to Databases

- Today: Relational databases; SQL
- Introduction to Microsoft Access
- Designing a Relational DB
- Building MS Access Applications

Outline: Databases

- The Relational Abstraction
  - Tables of data
  - Operations on tables
- Extracting data from Databases: Queries, SQL
- Computer Representation of Databases: Indexes
- DBMS
What is a Database

- An abstraction for storing and retrieving related pieces of data
- Many different kinds of databases have been proposed
  - hierarchical, network, etc.
  - each kind supports a different abstract model for organizing data
  - in this class, we will only explain relational databases
    - sets of tables of related data

Example DB: Fortune 500 Companies

- company
  - compname  sales  assets  netincome  empls  indcode  yr
  - allied  9115000  13271000  -279000  143800  37  85
  - boeing  9035000  7593000  292000  95700  37  82
- industry codes
  - indcode  indname
  - 42  pharmaceuticals
  - 44  computers
  - ...
The Relational Abstraction

- Information is in tables
  - Also called (base) relations
- Columns define attributes
  - Also called fields or domains
- Rows define records
  - Also called tuples
- Cells contain values
  - All cells in column have information of same type
    - e.g., integer, floating point, text, date

Operating on Databases: SQL

- Every abstraction needs an interface through which users invoke abstract operations
  - graphical interface
  - language
- Structured Query Language
- Has all those operations
- We'll focus only on queries
  - Query = question
  - Extract some data from one or more tables to answer a particular question
The Select Statement

- Every select statement yields a table of values as output
  - Sometimes there’s only one row in the table!

  ```
  select columns and/or expressions
  from tables
  where conditions on the rows
  group by group rows together
  having conditions on the groups
  order by order the rows
  into temp save results of query in a temporary table
  ```

Display Company Data

```sql
SELECT *
FROM company;
```
Choose Columns

- Choosing a subset of columns is sometimes called "project" operation
- Display company name and income for each year
  - `SELECT compname, netincome, yr`  
  - `FROM company;`

<table>
<thead>
<tr>
<th>compname</th>
<th>netincome</th>
<th>yr</th>
</tr>
</thead>
<tbody>
<tr>
<td>allied</td>
<td>-279000</td>
<td>85</td>
</tr>
<tr>
<td>boeing</td>
<td>292000</td>
<td>82</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Choose Rows

- Find performance data for 1984 for boeing
  - `SELECT compname, netincome, yr`  
    - `FROM company`  
    - `WHERE yr = 84 AND compname = "boeing";`
- Which companies lost money in 1984?
Compute Columns

- Find return on assets for each year
  
  SELECT compname, yr,  
  (netincome/assets) AS roa
  FROM company;

- Nice names for output columns
  - Name following computed column (e.g., roa) will be used to name output column

- Find company-years with roa of more than 15%

Sorting

- Can sort output by contents of a column
  - sort in ascending or descending order
  - sort by more than one column (second one breaks ties)

- Sort companies by 1984 profits
  
  SELECT compname, netincome
  FROM company
  WHERE yr = 84
  ORDER BY netincome DESC;

- Sort companies by 1984 return on assets
Aggregates

- Can make calculations on entire columns
  - sum, avg, max, min, count

- How many apparel companies are in database and what are their total sales for 1984?
  ```sql
  SELECT Count(*) AS number,
         Sum(sales) AS totalsales
  FROM company
  WHERE indcode = 40 and yr = 84;
  ```
  - returns a table with just one row!

- What is average percent roa for apparel companies in 1984?

Grouping and Aggregates

- Each different value for the group by fields defines a new group
- One row of output is produced for each group
- Several rows may belong to same group
  - Aggregate those using aggregation operator
- Compute total sales by all companies for each year
  ```sql
  SELECT yr,
         Sum(sales) AS totalsales
  FROM company
  GROUP BY yr;
  ```
More examples

- Compute total sales by all companies for each year
  
  ```sql
  SELECT yr, Sum(sales) AS totalsales
  FROM company
  GROUP BY yr;
  ```

- Compute total sales for each company

- What are the leading industries in total sales for 1984?

Joins

- Combine rows from one table with rows from another
- Usually join on some common column
  - Don’t combine rows unless their value in the common column is the same
  - Where clause says the common column must be same in each table
- Find the industry name for each company
  
  ```sql
  SELECT company.compname AS compname, codes.indname AS industry
  FROM company, codes
  WHERE company.indcode = codes.indcode;
  ```
Example DB: Fortune 500 Companies

- company

<table>
<thead>
<tr>
<th>compname</th>
<th>sales</th>
<th>assets</th>
<th>netincome</th>
<th>empls</th>
<th>indcode</th>
<th>yr</th>
</tr>
</thead>
<tbody>
<tr>
<td>allied</td>
<td>9115000</td>
<td>13271000</td>
<td>-279000</td>
<td>143800</td>
<td>37</td>
<td>85</td>
</tr>
<tr>
<td>boeing</td>
<td>9035000</td>
<td>7593000</td>
<td>292000</td>
<td>95700</td>
<td>37</td>
<td>82</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- industry codes

<table>
<thead>
<tr>
<th>indcode</th>
<th>indname</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>pharmaceuticals</td>
</tr>
<tr>
<td>44</td>
<td>computers</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Database Representations

- Requirements:
  - Minimize disk space taken by database
  - Enable fast retrieval of records with desired properties

- Main ideas:
  - Store tables as sequential files
    - within each table records can be stored in any order
  - Augment those tables with indexes to accelerate retrieval
Indexes

- Like a book index, it says where to find things
- Indexes can make queries run faster
- To understand indexes we have to break the abstraction barrier
  - How are tables stored?
    - Assume they are stored sequentially, one row after the other
      - Each row takes a fixed number of bytes of storage
  - How are queries processed?

Example

```
SELECT *
FROM company
WHERE indcode = 42;
```

- Suppose know nothing about order of records in company
  - check each record in sequence
    - if indcode = 42, put it into output table

- Suppose know records stored in ascending order by indcode
  - How will this help?

- So why not sort the records by indcode to make this query go faster?
Index on indcode

- Each index entry is an (indcode, record position) pair
- Leave the storage of the records alone
- Sort the index entries in increasing order by indcode
- Can store several indices (e.g., on indcode, on assets, etc.)
  - Requires less space than keeping several sorted copies of the actual tables
- Can we do even better?
  - Can we skip directly to index entries for indcode 42?

Linear vs. Tree Indices

- As described so far, linear access to index entries
  - 37, 7 - 37, 4 - 40, 6 - 42, 5 - 42, 8 - 42, 2 - 44, 4 - 45, 8 - 45, 2
- Tree access to index entries
- Now can avoid looking at first three index entries and last three
More on Trees

- That was a trinary tree
  - Each node had three “children”
  - Three is called the breadth
- Often use binary trees
  - Each node has ____ children
- Binary trees give fast access—if they are “balanced”
- If one “branch” of the tree is a lot heavier (has more nodes), lose benefit
- B-trees are trees that are guaranteed to stay pretty balanced, even as you add new nodes
- Most indices are implemented with some variation on B-trees

DBMS

- A Database Management System (DBMS) maintains the abstraction
  - Translates relational tables from/to internal representation
  - Implements operations on relational tables
    - Creates/Modifies tables
    - Inserts/Deletes data
    - Runs queries
    - ...
- Usually, DBMS builds on top of the OS-provided abstraction of files to store tables
  - this is an example of abstraction layering