CSC 443
Database Management Systems

Winter 2016
Professor: Marina Barsky
http://www.cdf.toronto.edu/~csc443h/winter/
Time of “Big Data”

• We aggressively acquire and keep data forever
• We feel real freedom when all data is available
• Implications for our live are enormous
• We use data for purposes different than it was primarily collected
## Data Units

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>Kilo</td>
<td>$2^{10}$</td>
<td>$10^3$</td>
</tr>
<tr>
<td>M</td>
<td>Mega</td>
<td>$2^{20}$</td>
<td>$10^6$</td>
</tr>
<tr>
<td>G</td>
<td>Giga</td>
<td>$2^{30}$</td>
<td>$10^9$</td>
</tr>
<tr>
<td>T</td>
<td>Tera</td>
<td>$2^{40}$</td>
<td>$10^{12}$</td>
</tr>
<tr>
<td>P</td>
<td>Peta</td>
<td>$2^{50}$</td>
<td>$10^{15}$</td>
</tr>
</tbody>
</table>
What is "Big Data"?

• Basic demographic information—age, sex, income, ethnicity, language, religion, housing status, and location—of every living human being on the planet can be stored in 100GB

• This would create a table of 6.75 billion rows and 10 columns.

• Should that be considered "big data"?

From “Pathologies of Big Data” Article by Adam Jacobs in the ACM Communications, August 2009.
Big-Data: Volume

• The web
  • 20+ billion web pages x 20KB = 400+ TB
  • One computer can read 30-35 MB/sec from one disk – 4 months just to read the web
Big-Data: **Variety**

- **NSF Ocean Observatories Initiative**
  - Data is collected from satellites, vessels, censors
  - 1000 km of optic cable on the seafloor with thousands of chemical, physical, biological sensors
  - 50 TB/year of different data types
Big-Data: \textbf{Velocity}

- Large Synoptic Survey Telescope (LSST)
  - 40 TB/day
  - 40+ PB in its 10 year lifetime
  - 400 mbps sustained data exchange rate between Chile and NSCA

- Largest database in the world: World Data Centre for Climate (WDCC):
  - 100 TB of sensor data/year
  - 110 TB of simulation data/year
  - 6PB of additional information stored on tapes
Big Data: 4V

- **Volume**
- **Variety**
- **Velocity**

- **Veracity**: can we trust this data?
Evolution of Science

• **Empirical Science** — collect and systematize facts

• **Theoretical Science** — formulate theories and empirically test them

• **Computational Science** — run automatic proofs, simulations

• **e-Science (Data Science)** — collect data without clear goal - and test theories, find patterns in the data itself
Science is about asking questions

Traditionally: “Query the world”
Data acquisition for a specific hypotheses

Data science: “Download the world”
Data acquired en masse in support of future hypotheses
“Big Data” - challenge

The cost of data acquisition has dropped

The cost of processing, integrating and analyzing data is the new bottleneck

“...the necessity of grappling with Big Data, and the desirability of unlocking the information hidden within it, is now a key theme in all the sciences – arguably the key scientific theme of our times”

F. Diebold
Efficient data manipulation

Poll: How much time modern scientists spend “handling data” as opposed to “doing science”?  
Mode answer: 90%

“... no greater barrier to effective data management will exist than the variety of incompatible data formats, non-aligned data structures, and inconsistent data semantics”

D. Laney, 2001

“Big Data ... and the Next Wave of InfraSress”

J. Mashey
Database Management Systems to the rescue

• Unified data model
• Efficient queries
• Fault-tolerance
• ...


Many facets of Database studies

• Logical design.
  • What kinds of information to store?
  • How to \textit{model} data?
  • How do data items connect?

• Database programming.
  • How does one express queries on the database?
  • How is database programming combined with conventional programming?

• Database system implementation.
  • How does one build a DBMS
What do we call a \textit{database}?

A \textit{collection of data} that exists over a long period of time, organized to afford efficient retrieval.

Two characteristics:

- Non-volatile reliable storage
- Organized for efficient operations
Useful definitions

• A *data model* is a collection of concepts for describing data.
  • Logical models: The *relational model of data* is the most widely used model today. Main concept: *relation*, basically a table with rows and columns.
  • Physical models

• A *schema* is a description of a particular collection of data, using a given data model.
  • Every relation has a *schema*, which describes the columns, or fields.

• A *view* – result of a stored query.
  • Same data – multiple views
Example: University Database

• Logical model:
  Relational: tables

• Schema:

  Students (sid: string, name: string, age: integer, gpa:real)
  Courses (cid: string, cname:string, credits:integer)
  Enrolled (sid:string, cid:string, grade:string)

• Physical model:
  Relations stored as unordered files.
  Index on first column of Students.

• View:
  Course_info (cid:string, enrollment:integer)
What is a *Database Management System (DBMS)*

A complex *software* for storing and managing databases.

Solves problems of:

- **Scale**: data exceeds main memory, specialized (quite complex) EM algorithms, efficiently implemented
- **Sharing**: using the same data by multiple user programs simultaneously
- **Fault-tolerance**: avoiding data loss
- **Consistency**: clean consistent snapshots of data, reinforcing data constraints
DBMS components

• The “cylindrical” component contains not only data, but also metadata, i.e. info about the structure of data.

• If DBMS is relational, metadata includes:
  – names of relations,
  – names of attributes of those relations, and
  – data types for those attributes (e.g., integer or character string).

• A database also maintains indexes for the data.
  – Indexes are part of the stored data.
  – Description of which attributes have indexes is part of the metadata.
DBMS provides:

1. Data independence:
   • not re-write the entire application if logical or physical data layouts have changed

2. Efficient access to data:
   • read, search, write, modify data quickly

3. Concurrent usage:
   • multiple user programs access the same data

4. Protection from improper use:
   • enforce constraints, permissions, data consistency

5. Fault-tolerance:
   • prevent loss of data from hardware failures

SQL - common high-level interface for schema definition (DDL) and for data queries and modification (DML)
1. Data Independence

One of the most important benefits of using a DBMS!

Applications are insulated from how data is structured and stored.

- **Logical data independence**: creating multiple views of the same data.
- **Physical data independence**: reasoning about data manipulation tasks independently of physical data layout.
2. Efficient access

• Smart data layouts
• Algorithms for external memory
• Data structures for external memory
• Query optimizer
3. Concurrent usage

• Why concurrent usage?
  • Efficient use of CPU and disk
  • Query execution may take time – do not make everyone else wait until it finishes

• Problems:
  • Interleaving actions of different user programs can lead to inconsistency: e.g., check is cleared while account balance is being computed.

• Vision:
  • Each user can pretend he is using a single-user system.
3. Concurrent usage: transactions

Transaction is an *atomic* sequence of database actions (reads/writes).

- Each transaction, executed completely, must leave the DB in a *consistent state* if DB is consistent when the transaction begins.
- All-or-nothing: if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. To the outside world, a committed transaction appears (by its effects on the database) to be indivisible ("atomic"), and an aborted transaction does not happen.
An atomic system guarantees atomicity in each and every situation, including power failures, errors, and crashes.

Idea: Keep a log (history) of all actions carried out by the DBMS while executing transactions:

- Before a change is made to the database, the corresponding log entry is forced to a safe location. *(WAL protocol; OS support for this is often inadequate.)*
- After a crash, the effects of partially executed transactions are undone using the log. *(Thanks to WAL, if log entry wasn’t saved before the crash, corresponding change was not applied to database!)*
3. Concurrent usage: two transactions on the same data

- Before reading/writing an object, a transaction requests a lock on the object, and waits till the DBMS gives it the lock. (2PL locking protocol.)
- All locks are released at the end of the transaction.
- If an action of T1 (say, writing X) affects T2 (which perhaps reads X), one of them, say T1, will obtain the lock on X first and T2 is forced to wait until T1 completes; this effectively orders the transactions.
- What if T2 already has a lock on Y and T1 requests a lock on Y to complete? (Deadlock!) T1 or T2 is aborted and restarted!
Course objectives

- Understand a Big-picture of different aspects of DBMS
- Experience challenges of database system implementation through programming assignments
- Learn techniques for working with big inputs
- Be able to solve system problems without reinventing the wheel – using what studied and understood
Relational DBMSs

Are they still around?
Data models - logical abstractions of data

- Files
- Network databases
- Hierarchical databases
- Relational databases
- Object-oriented databases
- NoSQL databases
- ...

• Insertion, updates, and deletion are complex and inefficient
• Lack of Data Independence: a change in structure demands a change in the application
• Unanticipated queries cannot be performed efficiently
History: hierarchical databases (1960-s - IBM IMS)

- Data is repetitively stored in many different entities.
- Slow search – scan entire model from top to bottom.
- One-to-many relationships only.
History: relational databases (1992)

God made the integers; all else is the work of man.
   L. Kronecker, 19-th century mathematician

Codd made relations; all else is the work of man.
   R. Ramakrishnan, author of your textbook

Edgar Codd (1923-2003)
Benefits of relational model

Think in terms of tables, not bits on disk.

“Activities of users at terminals should remain unaffected when the internal representation of data is changed.”

- Pre-relational: if your data changed, your application broke
- Early RDBMSs were buggy and slow, but required only 5% of the application code
Relational databases: key idea

Programs that manipulate tabular data exhibit an *algebraic structure* allowing reasoning and manipulation independently of physical data representation.
Algebraic optimization: symbolic reasoning on integers

\[ N = \frac{((z \times 2) + ((z \times 3) + 0))}{1} \]

Algebraic laws:

1. Identity: \( x + 0 = x \)
2. Identity: \( x / 1 = x \)
3. Distributive: \( ax + ay = a \times (x+y) \)
4. Commutative: \( x \times y = y \times x \)

Apply rules 1,3,4,2:
\[ N = (2+3) \times z \]

One operation instead of five, no division. 
*Closure*: each operation returns the value of the same type, so operations can be chained.

Same idea works with relational algebra!
Recap: algebra of tables

Selection $\sigma$

Projection $\pi$

Join $\bowtie$

Cross-product $\times$

Union $\cup$

Difference $-$

Intersection $\cap$
Relational algebra: quick question

Product (productID, name, price)
Customer (customerID, name, city)
Order (productID, customerID, store)

• What is the meaning of the following relational algebra query?

\[ \pi_{\text{name, store}} \sigma_{\text{city='Seattle'}} (\text{Orders} \bowtie \text{Customers}) \]

A. Produce list of stores where each customer from Seattle made orders

B. Produce all combinations of customers and stores in Seattle
Example: SQL query

Product (productID, name, price)
Customer (customerID, name, city)
Order (productID, customerID, store)

```
SELECT DISTINCT p.name, c.name
FROM Product p, Order o, Customer c
WHERE p.productID = o.productID
and c.customerID = o.customerID
and p.price > 100
and c.city = 'Seattle'
```
One SQL - many equivalent RA expressions

```
SELECT DISTINCT p.name, c.name
FROM Product p, Order o, Customer c
WHERE p.productID = o.productID and c.customerID = o.customerID
and p.price > 100 and c.city = ‘Seattle’
```

\[
\pi_{\text{p.name, c.name}} \sigma_{\text{p.price >100 and c.city = ‘Seattle’}} (P \times O \times C)
\]

\[
\pi_{\text{p.name, c.name}} \sigma_{\text{p.price >100 and c.city = ‘Seattle’}} ((P \bowtie O) \bowtie C)
\]

\[
\pi_{\text{p.name, c.name}} \sigma_{\text{p.price >100 and c.city = ‘Seattle’}} ((C \bowtie O) \bowtie P)
\]

\[
\pi_{\text{p.name, c.name}} (\sigma_{\text{price >100}} (P) \bowtie \sigma_{\text{c.city = ‘Seattle’}} (C)) \bowtie O)
\]
Symbolic reasoning on big tables: query plan 1

\[ \pi_{\text{p.name, c.name}} \sigma_{\text{p.price >100 and c.city = 'Seattle'}}((P \bowtie O) \bowtie C) \]

\[ \pi \]
\[ \left\{ \begin{array}{l}
\pi_{\text{p.name, c.name}} \\
\sigma_{\text{price >100 and city = 'Seattle'}} \\
\bowtie_{\text{customerID = customerID}} \\
\bowtie_{\text{productID = productID}}
\end{array} \right. \]

Product \hspace{1.5cm} Order \hspace{1.5cm} Customer
Symbolic reasoning on big tables: query plan 2

\[ \pi_{p.name, c.name} (\sigma_{\text{price} > 100} (P) \bowtie O) \bowtie (\sigma_{\text{c.city} = \text{Seattle}} (C))) \]

\[
\begin{align*}
\pi & \quad \text{p.name, c.name} \\
\bowtie & \quad \text{customerID} = \text{customerID} \\
\bowtie & \quad \text{productID} = \text{productID} \\
\sigma & \quad \text{price} > 100 \\
\sigma & \quad \text{city} = \text{Seattle}'
\end{align*}
\]

Pushing selections!
Quick question:
In what sense is "Algebraic Optimization" "optimizing" a user query?

A. The process uses faster algorithms to perform each step.

B. The expression is executed multiple times until the optimal result is determined.

C. The process finds an equivalent expression to the original, but one that is less expensive to compute - the expression has been "optimized".
Case in favor of Relational Database Management Systems

RDBMS provides:

• Physical and logical data independence
• Automatic indexing
• Efficient implementation of RA operators
• Query optimization
• Support and guarantees of atomic transactions

Imagine adding all these features yourself for your next data product!
When to use RDBMS

• Fast application development
• Data integrity and security is important
• Loss of data is unacceptable
• Concurrent data modification: by multiple users
• Data can be easily modeled as relations
When to consider alternative data stores

- String databases
- Audio, video databases
- Document databases (*Google Big Table*)
- Graph databases
Plan

• Physical storage
  • Memory hierarchy (RAM, HDD)
  • Using secondary storage effectively (EM algorithms, EM data structures - indexes)
  • Crash recovery (Redundant Arrays of Independent Disks)

• Query processing
  • Implementing relational algebra operators
  • Query optimization

• Parallel systems
  • Map-Reduce
  • Peer-to-peer

• Concurrency and recovery
  • Locking
  • Logging

• Alternative data stores
  • NoSQL
  • String databases
  • Graph databases