Course Goals

- Architectures of modern enterprise information systems
- Understand the functionality of modern database and data mining systems
- Understand where database systems and data mining fit into an enterprise information system
- Learn to ask the right questions
- Learn how to use several important tools
  - Data modeling (DeZign for Databases)
  - Data mining (SAS Enterprise Miner)
SAS Enterprise Miner

Course Outline

- 1/26 Database Management Systems
- 1/28 Enterprise Information Architectures
- 2/2, 2/4, and 2/9: Data Modeling
- 2/11, 2/16, 2/18, 2/23, and 2/25: Data Mining
- 3/2 OLAP
- 3/4 Web Services
- 3/9 Future Trends

Course Mechanics

- Temporary course homepage: [http://www.cs.cornell.edu/johannes/teaching/NBA518](http://www.cs.cornell.edu/johannes/teaching/NBA518)
- Slides will be online the morning before each lecture
- Readings for each class will be available online
- Office hours:
  - Tuesdays 1:30-2:30, Upson Hall 4105B
  - Mondays and Wednesdays from 2:00 – start of class in Sage Hall Atrium
- Always welcome to ask questions via email (johannes@cs.cornell.edu)
- Ask questions after the lecture
Grading

- Five homework assignments:
  - Enterprise architectures (15%)
  - Data modeling (15%)
  - Data mining I: Classification (15%)
  - Data mining II: Clustering and Associations (15%)
  - A complete case study (in groups of 2-3 students, 20%)

- Class participation (20%): Quality and not quantity counts

Introduction: About the Instructor

Johannes Gehrke is an Assistant Professor in the Department of Computer Science at Cornell University. He obtained his Ph.D. in computer science from the University of Wisconsin-Madison in 1998; his graduate studies were supported by a Fulbright fellowship and an IBM fellowship.

Johannes' research interests are in the areas of data mining, data stream processing, and distributed data management for sensor networks and peer-to-peer networks. Johannes has received a National Science Foundation Career Award, an Arthur P. Sloan Fellowship, an IBM Faculty Award, and the Computer Research and Development Association of Hawaii Raymond Chow Young Investigator Award. He is the author of numerous publications on data mining and database systems, and he co-authored the undergraduate textbook Database Management Systems (McGrawHill (2002), currently in its third edition), used at universities all over the world.

Johannes has served as Program Co-Chair of the 2001 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Tutorial Chair for the 2001 IEEE International Conference on Data Mining, Area Chair for the Twentieth International Conference on Machine Learning, co-Chair of the 2003 ACM SIGKDD Cup, and he is serving as Program co-Chair of the 2004 ACM SIGKDD Conference.

Johannes has given courses and tutorials on data mining and data stream processing at international conferences and on Wall Street, and he has extensive industry experience as technical advisor.

Introduction: Students
Goal of This Lecture

- Understand the basic functionality of a database system

The Big Picture

INTRANET, VPN

Main Memory Cache

DBMS

Data Warehouse Application Server

Application Server

INTRANET, VPN

WWW Site Visitor

Internal User

Public Web Server

Business Transaction Server

Why Database Systems?

Discuss with your neighbor: What functionality is required from database systems in the following application scenarios:

- eBay (www.ebay.com)
- Barnes and Noble (www.bn.com)
- General Motors (www.gm.com)
- The Protein Data Bank (http://www.rcsb.org/pdb)
- Sprint (www.sprint.com)
- Your cell phone
Why Store Data in a DBMS?

- Benefits
  - Transactions (concurrent data access, recovery from system crashes)
  - High-level abstractions for data access, manipulation, and administration
  - Data integrity and security
  - Performance and scalability

A Digress – What Is a Transaction?

The execution of a program that performs a function by accessing a database.

Examples:
- Reserve an airline seat. Buy an airline ticket.
- Withdraw money from an ATM.
- Verify a credit card sale.
- Order an item from an Internet retailer.
- Download a video clip and pay for it.
- Play a bid at an on-line auction.

Transactions

- A transaction is an atomic sequence of actions
- Each transaction must leave the system in a consistent state (if system is consistent when the transaction starts).
- The ACID Properties:
  - Atomicity
  - Consistency
  - Isolation
  - Durability
Example Transaction: Online Store

Your purchase transaction:
- Atomicity: Either the complete purchase happens, or nothing
- Consistency: The inventory and internal accounts are updated correctly
- Isolation: It does not matter whether other customers are also currently making a purchase
- Durability: Once you have received the order confirmation number, your order information is permanent, even if the site crashes

Transactions (Contd.)

A transaction will commit after completing all its actions, or it could abort (or be aborted by the DBMS) after executing some actions.

Example Transaction: ATM

You withdraw money from the ATM machine
- Atomicity
- Consistency
- Isolation
- Durability

Commit versus Abort?
What are reasons for commit or abort?
Transactions: Examples

Give examples of transactions in the following applications. Which of the ACID properties are needed?

- EBay (www.ebay.com)
- Barnes and Noble (www.bn.com)
- General Motors (www.gm.com)
- The Protein Data Bank (http://www.rcsb.org/pdb)
- Sprint (www.sprint.com)
- Your cell phone

What Makes Transaction Processing Hard

- Reliability - system should rarely fail
- Availability - system must be up all the time
- Response time - within 1-2 seconds
- Throughput - thousands of transactions/second
- Scalability - start small, ramp up to Internet-scale
- Security – for confidentiality and high finance
- Configurability - for above requirements + low cost
- Atomicity - no partial results
- Durability - a transaction is a legal contract
- Distribution - of users and data

Reliability and Availability

- Reliability - system should rarely fail
- Availability - system must be up all the time

<table>
<thead>
<tr>
<th>Downtime</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 hour/day</td>
<td>95.8%</td>
</tr>
<tr>
<td>1 hour/week</td>
<td>99.41%</td>
</tr>
<tr>
<td>1 hour/month</td>
<td>99.86%</td>
</tr>
<tr>
<td>1 hour/year</td>
<td>99.9886%</td>
</tr>
<tr>
<td>1 minute/day</td>
<td>99.9988%</td>
</tr>
<tr>
<td>1 hour/20years</td>
<td>99.99942%</td>
</tr>
<tr>
<td>1 minute/week</td>
<td>99.99983%</td>
</tr>
</tbody>
</table>
Performance

• Response time - within 1-2 seconds
• Throughput - thousands of transactions/second
• Scalability - start small, ramp up to Internet-scale

What Makes TP Important?

• It is at the core of electronic commerce
• Most medium-to-large businesses use TP for their production systems. The business can’t operate without it.
• It is a huge slice of the computer system market — over $50B/year. Probably the single largest application of computers.

TP System Infrastructure

• User’s viewpoint
  • Enter a request from a browser or other display device
  • The system performs some application-specific work, which includes database accesses
  • Receive a reply (usually, but not always)
• The TP system ensures that each transaction
  • is an independent unit of work
  • executes exactly once, and
  • produces permanent results.
• TP system makes it easy to program transactions
• TP system has tools to make it easy to manage
TP System Infrastructure

End-User  

Presentation Manager  
requests  

Workflow Control  
(routes requests)  

Transaction Program  

Database System  

Front-End  
(Client)  

Back-End  
(Server)

System Characteristics

- Typically < 100 transaction types per application
- Transaction size has high variance. Typically,
  - 0-30 disk accesses
  - 10K - 1M instructions executed
  - 2-20 messages
- A large-scale example: airline reservations
  - 150,000 active display devices
  - plus indirect access via Internet travel agents
  - thousands of disk drives
  - 3000 transactions per second, peak

Exercise

- Reliability - system should rarely fail
- Availability - system must be up all the time
- Response time - within 1-2 seconds
- Throughput - thousands of transactions/second
- Scalability - start small, ramp up to Internet-scale
- Security - for confidentiality and high finance
- Configurability - for above requirements + low cost
- Atomicity - no partial results
- Durability - a transaction is a legal contract
- Distribution - of users and data

- Question: Think of a TP System that you know of, and discuss with your neighbor the requirements above. Contrast this with requirements of
  - The Cornell Electronic Directory
  - amazon.com
Concurrency Control for Isolation

(Start: A=$100; B=$100)

Consider two transactions:
• T1: START, A=A+100, B=B-100, COMMIT
• T2: START, A=1.06*A, B=1.06*B, COMMIT

The first transaction is transferring $100 from B's account to A's account. The second transaction is crediting both accounts with a 6% interest payment.

Database systems try to do as many operations concurrently as possible, to increase performance.

Example (Contd.)

(Start: A=$100; B=$100)

• Consider a possible interleaving (schedule):
  T1: A=A+$100, B=B-$100 COMMIT
  T2: A=1.06*A, B=1.06*B COMMIT
  End result: A=$106; B=$0

• Another possible interleaving:
  T1: A=A+100, B=B-100 COMMIT
  T2: A=1.06*A, B=1.06*B COMMIT
  End result: A=$112; B=$6

The second interleaving is incorrect! Concurrency control of a database system makes sure that the second schedule does not happen.

Ensuring Atomicity

• DBMS ensures atomicity (all-or-nothing property) even if the system crashes in the middle of a transaction.
• Idea: Keep a log (history) of all actions carried out by the DBMS while executing:
  • Before a change is made to the database, the corresponding log entry is forced to a safe location.
  • After a crash, the effects of partially executed transactions are undone using the log.
Recovery

- A DBMS logs all elementary events on stable storage. This data is called the log.
- The log contains everything that changes data: Inserts, updates, and deletes.
- Reasons for logging:
  - Need to UNDO transactions
  - Recover from a systems crash

Recovery: Example

(Simplified process)

- Insert customer data into the database
- Check order availability
- Insert order data into the database
- Write recovery data (the log) to stable storage
- Return order confirmation number to the customer

Why Store Data in a DBMS?

- Benefits
  - Transactions (concurrent data access, recovery from system crashes)
  - High-level abstractions for data access, manipulation, and administration
  - Data integrity and security
  - Performance and scalability
### Data Model

- A **data model** is a collection of concepts for describing data.
- Examples:
  - ER model (used for conceptual modeling)
  - Relational model, object-oriented model, object-relational model (actually implemented in current DBMS)

### The Relational Data Model

A relational database is a set of relations. Turing Award (Nobel Price in CS) for Codd in 1980 for his work on the relational model

- Example relation:
  ```
  Customers(cid: integer, name: string, byear: integer, state: string)
  ```

<table>
<thead>
<tr>
<th>cid</th>
<th>name</th>
<th>byear</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jones</td>
<td>1960</td>
<td>NY</td>
</tr>
<tr>
<td>2</td>
<td>Smith</td>
<td>1974</td>
<td>CA</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>1950</td>
<td>NY</td>
</tr>
</tbody>
</table>

### The Relational Model: Terminology

- Relation instance and schema
- Field (column)
- Record or tuple (row)
- Cardinality

<table>
<thead>
<tr>
<th>cid</th>
<th>name</th>
<th>byear</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jones</td>
<td>1960</td>
<td>NY</td>
</tr>
<tr>
<td>2</td>
<td>Smith</td>
<td>1974</td>
<td>CA</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>1950</td>
<td>NY</td>
</tr>
</tbody>
</table>
Customer Relation (Contd.)

• In your enterprise, you are more likely to have a schema similar to the following:

Customers(cid, identifier, nameType, salutation, 
firstName, middleNames, lastName, 
culturalGreetingStyle, gender, customerType, degrees, 
education, company, department, job, 
primaryPhone, primaryFax, email, website, building, 
floor, mailstop, addressType, streetNumber, 
streetName, streetDirection, POBox, city, state, zipCode, 
region, country, assembledAddressBlock, currency, 
martialStatus, bYear, profession)

Product Relation

• Relation schema:
  Products(pid: integer, pname: string, price: float, 
category: string)

• Relation instance:

<table>
<thead>
<tr>
<th>pid</th>
<th>pname</th>
<th>price</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Intel PIII-700</td>
<td>300.00</td>
<td>hardware</td>
</tr>
<tr>
<td>2</td>
<td>MS Office Pro</td>
<td>500.00</td>
<td>software</td>
</tr>
<tr>
<td>3</td>
<td>IBM DB2</td>
<td>5000.00</td>
<td>software</td>
</tr>
<tr>
<td>4</td>
<td>Thinkpad 600E</td>
<td>5000.00</td>
<td>hardware</td>
</tr>
</tbody>
</table>

Transaction Relation

• Relation schema:
  Transactions(tid: integer, 
tdate: date, 
cid: integer, 
pid: integer)

• Relation instance:

<table>
<thead>
<tr>
<th>tid</th>
<th>tdate</th>
<th>cid</th>
<th>pid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/1/2000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1/1/2000</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1/1/2000</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2/1/2000</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2/1/2000</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
The Relational DBMS Market

Market Share Of RDBMS in 1998
(Gartner Group 4/1999)

0.0%
10.0%
20.0%
30.0%
40.0%
50.0%
60.0%
70.0%

Oracle 7/8 and Lite
Microsoft SQL
IBM DB2
Sybase
Informix
NCR
Other

The Relational DBMS Market (Contd.)

Best-Selling Client/Server DBMS
(Computer Reseller News 8/1999)

0%
10%
20%
30%
40%
50%

Microsoft
Oracle
Sybase
IBM
Others

The Relational DBMS Market (Contd.)

Market Share Of Database Revenues
(Computer Reseller News, 5/1999)

0.0%
5.0%
10.0%
15.0%
20.0%
25.0%
30.0%
35.0%

IBM
Oracle
Microsoft
Informix
Sybase
Other
The Object-Oriented Data Model

- Richer data model. Goal: Bridge impedance mismatch between programming languages and the database system.
- Example components of the data model: Relationships between objects directly as pointers.
- Result: Can store abstract data types directly in the DBMS
  - Pictures
  - Geographic coordinates
  - Movies
  - CAD objects

Object-Oriented DBMS

- Advantages: Engineering applications (CAD and CAM and CASE computer aided software engineering), multimedia applications.
- Disadvantages:
  - Technology not as mature as relational DBMS
  - Not suitable for decision support, weak security
  - Vendors are much smaller companies and their financial stability is questionable.

Object-Oriented DBMS (Contd.)

Vendors:
- Gemstone ([www.gemstone.com](http://www.gemstone.com))
- Objectivity ([www.objiv.com](http://www.objiv.com))
- ObjectStore ([www.objectstore.net](http://www.objectstore.net))
- POET ([www.poet.com](http://www.poet.com))
- Versant ([www.versant.com](http://www.versant.com), merged with POET)

Organizations:
The OO DBMS Market

Object-Relational DBMS

- Mixture between the object-oriented and the object-relational data model
  - Combines ease of querying with ability to store abstract data types
  - Conceptually, the relational model, but every field
- All major relational vendors are currently extending their relational DBMS to the object-relational model

Query Languages

We need a high-level language to describe and manipulate the data

Requirements:
- Precise semantics
- Easy integration into applications written in C++/Java/Visual Basic/etc.
- Easy to learn
- DBMS needs to be able to efficiently evaluate queries written in the language
Relational Query Languages

- The relational model supports simple, powerful querying of data.
  - Precise semantics for relational queries
  - Efficient execution of queries by the DBMS
  - Independent of physical storage

SQL: Structured Query Language

- Developed by IBM (System R) in the 1970s
- ANSI standard since 1986:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)
- More about SQL in the next lecture

Example Query

- Example Schema:
  Customers
    cid: integer,
    name: string,
    byear: integer,
    state: string
- Query:
  SELECT
    Customers.cid,
    Customers.name,
    Customers.byear,
    Customers.state
  FROM Customers
  WHERE Customers.cid = 3

<table>
<thead>
<tr>
<th>cid</th>
<th>name</th>
<th>byear</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jones</td>
<td>1960</td>
<td>NY</td>
</tr>
<tr>
<td>2</td>
<td>Smith</td>
<td>1974</td>
<td>CA</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>1950</td>
<td>NY</td>
</tr>
</tbody>
</table>
Example Query

```sql
SELECT Customers.cid,
       Customers.name,
       Customers.byear,
       Customers.state
FROM Customers
WHERE Customers.cid = 1
```

<table>
<thead>
<tr>
<th>cid</th>
<th>name</th>
<th>byear</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jones</td>
<td>1960</td>
<td>NY</td>
</tr>
<tr>
<td>2</td>
<td>Smith</td>
<td>1974</td>
<td>CA</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>1950</td>
<td>NY</td>
</tr>
</tbody>
</table>

Why Store Data in a DBMS?

- Benefits
  - Transactions (concurrent data access, recovery from system crashes)
  - High-level abstractions for data access, manipulation, and administration
  - Data integrity and security
  - Performance and scalability

Integrity Constraints

- Integrity Constraints (ICs): Condition that must be true for any instance of the database.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
  - A legal instance of a relation is one that satisfies all specified ICs.
  - DBMS should only allow legal instances.
- Example: Domain constraints.
Primary Key Constraints

- A set of fields is a superkey for a relation if no two distinct tuples can have same values in all key fields.
- A set of fields is a key if the set is a superkey, and none of its subsets is a superkey.
- Example:
  - \{cid, name\} is a superkey for Customers
  - \{cid\} is a key for Customers
- Where do primary key constraints come from?

Primary Key Constraints (Contd.)

- Can there be more than one key for a relation?
- What is the maximum number of superkeys for a relation?
- What is the primary key of the Products relation? How about the Transactions relation?

Foreign Keys, Referential Integrity

- Foreign key: Set of fields in one relation that refers to a unique tuple in another relation. (The foreign key must be a superkey of the second relation.)
- Example: The field cid in the Transactions relation is a foreign key referring to Customers.
- If all foreign key constraints are enforced, we say that referential integrity is achieved.
  - No dangling references.
  - Compare to links in HTML.
Foreign Keys: Example

- The pid field of the Transactions relation refers to the cid field of the Customer relation.

<table>
<thead>
<tr>
<th>tid</th>
<th>tdate</th>
<th>cid</th>
<th>pid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/1/2000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1/1/2000</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1/1/2000</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2/1/2000</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2/1/2000</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>name</th>
<th>byear</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jones</td>
<td>1960</td>
<td>NY</td>
</tr>
<tr>
<td>2</td>
<td>Smith</td>
<td>1950</td>
<td>NY</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>1974</td>
<td>CA</td>
</tr>
</tbody>
</table>

Enforcing Referential Integrity

- What should be done if a Transaction tuple with a non-existent Customer id is inserted? (Reject it!)
- What should be done if a Customer tuple is deleted?
  - Also delete all Transaction tuples that refer to it.
  - Disallow deletion of a Customer tuple that has associated Transactions.
  - Set cid in Transactions tuples to a default or special cid.
- SQL supports all three choices

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know state cannot be a key, but the assertion that cid is a key is given to us.
- Key and foreign key ICs are very common; a DBMS supports more general ICs.
Security

- **Secrecy:** Users should not be able to see things they are not supposed to.
  - E.g., A student can't see other students’ grades.
- **Integrity:** Users should not be able to modify things they are not supposed to.
  - E.g., Only instructors can assign grades.
- **Availability:** Users should be able to see and modify things they are allowed to.

Discretionary Access Control

- Based on the concept of access rights or privileges for objects (tables and views), and mechanisms for giving users privileges (and revoking privileges).
- Creator of data automatically gets all privileges on it.
  - DBMS keeps track of who subsequently gains and loses privileges, and ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed.
- Users can grant and revoke privileges

Role-Based Authorization

- In SQL-92, privileges are actually assigned to authorization ids, which can denote a single user or a group of users.
- In SQL:1999 (and in many current systems), privileges are assigned to roles.
  - Roles can then be granted to users and to other roles.
  - Reflects how real organizations work.
  - Illustrates how standards often catch up with "de facto" standards embodied in popular systems.
Security Is Important

Financial estimate of database losses, by activity in 1998

<table>
<thead>
<tr>
<th>Activity</th>
<th>Loss in million $</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theft of Proprietary Information</td>
<td>28.51</td>
</tr>
<tr>
<td>System Penetration by Outsiders</td>
<td>13.39</td>
</tr>
<tr>
<td>Financial Fraud</td>
<td>11.24</td>
</tr>
<tr>
<td>Unauthorized Insider Access</td>
<td>50.57</td>
</tr>
<tr>
<td>Laptop Theft</td>
<td>0.16</td>
</tr>
<tr>
<td>Total</td>
<td>103.87</td>
</tr>
</tbody>
</table>

Why Store Data in a DBMS?

• Benefits
  • Transactions (concurrent data access, recovery from system crashes)
  • High-level abstractions for data access, manipulation, and administration
  • Data integrity and security
  • Performance and scalability

DBMS and Performance

• Efficient implementation of all database operations
• Indexes: Auxiliary structures that allow fast access to the portion of data that a query is about
• Smart buffer management
• Query optimization: Finds the best way to execute a query
• Automatic high-performance concurrent query execution, query parallelization
Summary Of DBMS Benefits

- Transactions
  - ACID properties, concurrency control, recovery
- High-level abstractions for data access
  - Data models
- Data integrity and security
  - Key constraints, foreign key constraints, access control
- Performance and scalability
  - Parallel DBMS, distributed DBMS, performance tuning

The Big Picture (Revisited)

Next Lecture

- More about the components outside the database system:
  - Three-tier architectures
  - Application servers
  - Middleware

- Readings for next lecture will be online tomorrow morning, slides will be online Wednesday morning