

CIS 330: Applied Database Systems

Lecture 1: Introduction
Johannes Gehrke
johannes@cs.cornell.edu
http://www.cs.cornell.edu/johannes

Course Goals

• Understand the functionality of modern database systems
• Understand where database systems fit into an enterprise data management infrastructure
• Design and build data-driven applications websites

• Learn several important tools:
  • Database System: Microsoft SQL Server
  • Application Server: Apache Tomcat
  • Data Modeling tool: DeZign for Databases
• Learn several important technologies
  • JDBC, JSP, Servlets, XML/XSLT/XPath, web services, J2EE

Instructor

• Johannes Gehrke
  • http://www.cs.cornell.edu/johannes

• Office hours:
  • Tuesdays, 1:30-2:30, Upson Hall 4105B
  • Always welcome to ask questions via email (johannes@cs.cornell.edu)
  • Ask questions after the lecture
Course Mechanics

- Homepage will have all the relevant material
- Slides will be online before each lecture
- Every student who is enrolled in the class will receive a free loaner laptop
  - Laptops will be distributed this Thursday, January 29, from 5:30-6:30pm in B17 Upson Hall
  - You need to enroll by Thursday in order to receive a laptop!
- Course Outline: See Handout
- Please put up nametags

Software: DeZign for Databases

Prerequisites and Grading

- **Prerequisites:**
  - CS211; if you don't have CS211, talk to me after class

- **Grading:**
  - 20 (smaller and larger) homework assignments (no groups), total of 60%. See handout.
  - Two exams:
    - Midterm: 15%
    - Final: 20%
  - Class participation: 5%
This Lecture

- Three-tier architectures
- Introduction to database systems

The Big Picture

Enterprise Architectures

Three separate types of functionality:
- Data management
- Application logic
- Presentation

- The system architecture determines whether these three components reside on a single system ("tier") or are distributed across several tiers
Single-Tier Architectures

- All functionality combined into a single tier, usually on a mainframe
  - User access through dumb terminals

- Advantages:
  - Easy maintenance and administration
- Disadvantages:
  - Today, users expect graphical user interfaces.
  - Centralized computation of all of them is too much for a central system

Client-Server Architectures

- Work division: Thin client
  - Client implements only the graphical user interface
  - Server implements business logic and data management
- Work division: Thick client
  - Client implements both the graphical user interface and the business logic
  - Server implements data management

Client-Server Architectures (Contd.)

- Disadvantages of thick clients
  - No central place to update the business logic
  - Security issues: Server needs to trust clients
    - Access control and authentication needs to be managed at the server
    - Clients need to leave server database in consistent state
    - One possibility: Encapsulate all database access into stored procedures
  - Does not scale to more than several 100s of clients
    - Large data transfer between server and client
    - More than one server creates a problem: \( x \) clients, \( y \) servers:
      \( x^2y \) connections
The Three-Tier Architecture

Presentation tier
- Client Program (Web Browser)

Middle tier
- Application Server

Data management tier
- Database System

The Three Layers

- Presentation tier
  - Primary interface to the user
  - Needs to adapt to different display devices (PC, PDA, cell phone, voice access?)

- Middle tier
  - Implements business logic (implements complex actions, maintains state between different steps of a workflow)
  - Accesses different data management systems

- Data management tier
  - One or more standard database management systems

Example 1: Airline reservations

- Build a system for making airline reservations
- What is done in the different tiers?
  - Database System
    - Airline info, available seats, customer info, etc.
  - Application Server
    - Logic to make reservations, cancel reservations, add new airlines, etc.
  - Client Program
    - Log in different users, display forms and human-readable output
Example 2: Course Enrollment

- Build a system using which students can enroll in courses
- Database System
  - Student info, course info, instructor info, course availability, pre-requisites, etc.
- Application Server
  - Logic to add a course, drop a course, create a new course, etc.
- Client Program
  - Log in different users (students, staff, faculty), display forms and human-readable output

Three-Tier Architecture: Advantages

- Heterogeneous systems
  - Tiers can be independently maintained, modified, and replaced
- Thin clients
  - Only presentation layer at clients (web browsers)
- Integrated data access
  - Several database systems can be handled transparently at the middle tier
  - Central management of connections
- Scalability
  - Replication at middle tier permits scalability of business logic
- Software development
  - Code for business logic is centralized
  - Interaction between tiers through well-defined APIs: Can reuse standard components at each tier

Technologies

- Client Program
  - (Web Browser)
  - HTML
  - Javascript
  - XSLT
- Application Server
  - (Tomcat, Apache)
  - SQL, JSP, Servlets
  - Cookies, EJB, XPath, web services
- Database System
  - (Microsoft SQL Server)
  - XML, Stored Procedures
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

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

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, ethnicity, companyName, departmentName, jobTitle, primaryPhone, primaryFax, email, website, building, floor, mailstop, addressType, streetName, streetDirection, POBox, city, state, zipCode, region, country, assembledAddressBlock, currency, maritalStatus, 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>3</td>
<td>2/1/2000</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

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)
- Objectivity (www.objy.com)
- ObjectStore (www.objectstore.net)
- POET (www.poet.com)
- Versant (www.versant.com, merged with POET)

Organizations:
- OMG: Object Management Group (www.omg.org)

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>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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>name</th>
<th>byear</th>
<th>state</th>
</tr>
</thead>
<tbody>
<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 with k fields?
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.

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