Διαδικτυακές Βάσεις Δεδομένων

Εισαγωγή

1. Περιεχόμενο του μαθήματος
2. Βάσεις δεδομένων (σε 1 ώρα !)
3. Εισαγωγή: Ο παγκόσμιος ιστός και βάσεις δεδομένων

Αντικείμενο του Μαθήματος

Σύγχρονα θέματα βάσεων δεδομένων - "διαδικτυακές βάσεις δεδομένων"

The network is the computer

- Web
- pervasive computing

Αντικείμενο του Μαθήματος

- Νέα *μοντέλα δεδομένων* (στη θέση των σχεσιακών, αντικειμενοστραφών, κλπ)
- Καινούργιες *αρχιτεκτονικές* (στη θέση των κεντρικών, πελάτη/εξυπηρέτη, κλπ)
- Νέοι τρόποι *εξόρυξης* πληροφορίας (πέρα από την SQL και τις μηχανές αναζήτησης)
- Νέοι τρόποι *μετάδοσης* αυτής της πληροφορίας

Φύση του Μαθήματος

Undergraduate class

Professors teach, students *have to* learn

Graduate core class

Professors teach, students *want to* learn

Graduate reading class

Students teach, professors want to learn

Θέματα αυτή τη χρονιά

δανασμένο από διαφάνειες του M. Carey για το μάθημα: "Post-Modern Database Systems: Databases Meet the Web", Berkeley 1998-99

Φύση του Μαθήματος

Ασκήσεις (20%)

κατανόηση ύλης

Τελικό διαγώνισμα (15%)

ανακεφαλαίωση

Βιβλιογραφική εργασία (25%)

συμμετοχή στη διδασκαλία/μάθημα

Εργασία (30%)

στο θέμα της βιβλιογραφικής εργασίας ("learn by doing")

Παρουσιάσεις (10%)

Συστήματα Διαχείρισης Βάσεων Δεδομένων

Διαφάνειες του J. Hellerstein (Univ. of Berkeley)

Το πλήρες set στο web site

Επίσης στη σελίδα μου διαφάνειες από το προπτυχιακό

Databases on 1 Foot

Joe Hellerstein
Computer Science Division
UC Berkeley

Overview for the day

- Motivation
- Relational model
- Relational Algebra
- SQL
- Concurrency Control
- Recovery

More Background

- Overview Texts:
  - Ramakrishnan, *Database Management Systems* (the cow book)
  - Silberschatz, Korth, Sudarshan: *Database System Concepts* (the sailboat book)
  - O'Neil: *Database Principles, Programming, Performance*
  - Ullman & Widom: *A 1st Course in Database Systems*
  - Garcia-Molina, Ullman & Widom: *Database Implementation*
- Graduate-level Texts
  - Stonebraker & Hellerstein, eds. *Readings in Database Systems* (http://redbook.cs.berkeley.edu)
  - Gray & Reuter: *Transaction Processing: Concepts and Techniques.*
  - Ullman: *Principles of Database Systems*

More Background

- Πληροφορία στο web:
  - hhttp://www.informatik.uni-trier.de/~ley/db/index.html
- ACM Sigmod
- IEEE TCDE
- Περιοδικά: ACM TODS, IEEE TKDE, Information Systems, VLDB Journal, …
- Συνέδρια: ACM SIGMOD, ACM PODS, VLDB, IEEE ICDE, VLDB, EDBT, IEEE ICDCS, …
- dbworld mailing list

Database Management Systems

- What more could we want than a file system?
  - Simple, efficient ad hoc[1] queries
  - concurrency control
  - recovery
  - benefits of good data modeling

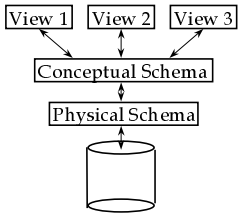
[1] ad hoc: formed or used for specific or immediate problems or needs

Describing Data: Data Models

- A **data model** is a collection of concepts for describing data.
- A **schema** is a description of a particular collection of data, using the a given data model.
- The **relational model of data** is the most widely used model today.
  - Main concept: **relation**, basically a table with rows and columns.
  - Every relation has a **schema**, which describes the columns, or fields.

Levels of Abstraction

- *Physical schema* describes the files and indexes used.
- *Conceptual schema* defines logical structure
- *Views* describe how users see the data.
- Many views, single conceptual (logical) schema and physical schema.



Example: University Database

- Conceptual schema:
  - *Students(sid: string, name: string, login: string, age: integer, gpa:real)*
  - *Courses(cid: string, cname:string, credits:integer)*
  - *Enrolled(sid:string, cid:string, grade:string)*
- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.
- External Schema (View):
  - *Course_info(cid:string,enrollment:integer)*

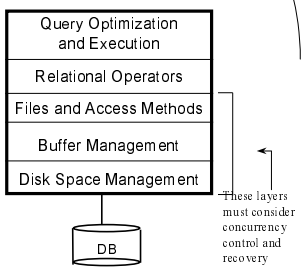
Data Independence

- Applications insulated from how data is structured and stored.
- **Logical data independence:** Protection from changes in *logical* structure of data.
- **Physical data independence:** Protection from changes in *physical* structure of data.

☛ *One of the most important benefits of using a DBMS!*

Structure of a DBMS

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components.
- This is one of several possible architectures; each system has its own variations.

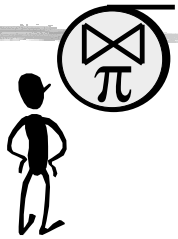


Advantages of a DBMS

- Data independence
- Efficient data access
- Data integrity & security
- Data administration
- Concurrent access, crash recovery
- Reduced application development time
- So why not use them always?
  - Can be expensive, complicated to set up and maintain
  - This cost & complexity must be offset by need

- Often worth it!

Relational Algebra



By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and, in effect, *increases the mental power of the race.*

-- **Alfred North Whitehead (1861 - 1947)**

Relational Query Languages

- **Query languages**: Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages **!=** programming languages!
  - QLs not expected to be "Turing complete".
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Formal Relational Query Languages

Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:

1. **Relational Algebra:** More operational, very useful for representing internal execution plans. (Database byte-code...)
2. **Relational Calculus:** Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative* -- SQL comes from here.)

Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance.
  - *Schemas* of input relations for a query are fixed (but query will run regardless of instance!)
  - The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.
  - Languages are *closed* (can compose queries)

Example Instances

- "Sailors" and "Reserves" relations for our examples.
- We'll use positional or named field notation, assume that names of fields in query results are 'inherited' from names of fields in query input relations.

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Relational Algebra

Relational Algebra

- Basic operations:
  - **Selection** (σ) Selects a subset of rows from relation.
  - **Projection** (π) 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** ($\cup$) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - Intersection, **join**, division, renaming: Not essential, but (very!) useful.

Projection

- Deletes attributes that are not in *projection list.*
- *Schema* of result:
  - 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?)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname,rating}(S2)$

age
35.0
55.5

$\pi_{age}(S2)$

Selection

- Selects rows that satisfy *selection condition.*
- No duplicates in result!
- *Schema* of result:
  - identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$\sigma_{rating>8}(S2)$

sname	rating
yuppy	9
rusty	10

$\pi_{sname,rating}(\sigma_{rating>8}(S2))$

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?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

sid	sname	rating	age
22	dustin	7	45.0

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

Cross-Product

- S1 x 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.
  - *Conflict*: Both S1 and R1 have a field called *sid.*

Cross-Product

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

☛ *Renaming operator*:

Joins

- **Condition Join:**

$$R \bowtie_{} S = \sigma_{} (R \times S)$$

- *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.*

Joins

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

Joins

- *Equi-Join*: Special case: condition *c* contains only conjunction of ***equalities***. $S1 \bowtie_{sid} R1$
- *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- *Natural Join*: Equijoin on *all* common fields.

Joins

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{sid} R1$$

Basic SQL Query

SELECT	[DISTINCT] *target-list*
FROM	*relation-list*
WHERE	*qualification*

Basic SQL Query

- **relation-list** : A list of relation names
  - possibly with a *range-variable* after each name
- **target-list** : A list of attributes of tables in *relation-list*
- **qualification :** Comparisons combined using AND, OR and NOT.
  - Comparisons are Attr *op* const or Attr1 *op* Attr2, where *op* is one of

$$<, >, =, \leq, \geq, \neq$$

- DISTINCT: optional keyword indicating that the answer should not contain duplicates.
  - Default is that duplicates are *not* eliminated!

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they fail *qualifications*.
  - Delete attributes that are not in *target-list*.
  - If DISTINCT is specified, eliminate duplicate rows.
- Probably the least efficient way to compute a query!
  - An optimizer will find more efficient strategies *same answers*.

Example of Conceptual Evaluation

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

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Aggregate Operators

- Significant extension of relational algebra.

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

single column

```
SELECT COUNT (*)
FROM Sailors S
```

```
SELECT AVG (S.age)
FROM Sailors S
WHERE S.rating=10
```


GROUP BY and HAVING

- So far, we've applied aggregate operators to all (qualifying) tuples.
  - Sometimes, we want to apply them to each of several *groups* of tuples.
- Consider: *Find the age of the youngest sailor for each rating level.*
  - In general, we don't know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):

For i = 1, 2, ... , 10:

```
SELECT MIN (S.age)
FROM Sailors S
WHERE S.rating = i
```


Queries With GROUP BY and HAVING

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

- The *target-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
  - The attribute list (i) must be a subset of *grouping-list*. Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group. (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

Null Values

- Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., no spouse's name).
  - SQL provides a special value *null* for such situations.
- The presence of *null* complicates many issues. E.g.:
  - Special operators needed to check if value is/is not *null*.
  - Is *rating>8* true or false when *rating* is equal to *null*? What about AND, OR and NOT connectives?
  - We need a 3-valued logic (true, false and *unknown*).
  - Meaning of constructs must be defined carefully. (e.g., WHERE clause eliminates rows that don't evaluate to true.)
  - New operators (in particular, *outer joins*) possible/needed.

Query Optimization & Processing

- Optimizer maps SQL to algebra tree with specific algorithms
  - access methods and join algorithms
- relational operators implemented as *iterators*
  - open()
  - next(possible with condition)
  - close
- processing engine is a pull-based, single-threaded data flow
  - parallelizes naturally

Embedded SQL

- SQL commands can be called from within a host language (e.g., C or COBOL) program.
  - SQL statements can refer to host variables (including special variables used to return status).
  - Must include a statement to *connect* to the right database.
  - Requires compiler preprocessing
  - SQL relations are (multi-) sets of records, with no *a priori* bound on the number of records. No such data structure in C.
  - SQL supports a mechanism called a *cursor* to handle this.

Cursors

- Can declare a cursor on a relation or query statement (which generates a relation).
- Can *open* a cursor, and repeatedly *fetch* a tuple then *move* the cursor, until all tuples have been retrieved.

Cursor that gets names of sailors who've reserved a red boat, in alphabetical order

```
DECLARE sinfo CURSOR FOR
        SELECT S.sname
        FROM Sailors S, Boats B, Reserves R
        WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
        ORDER BY S.sname;

FETCH 5 IN sinfo;
```


Database APIs: alternative to embedding

- Rather than modify compiler, add library with database calls (API)
  - special procedures/objects
  - passes SQL strings from language, presents result sets in a language-friendly way
  - Microsoft's *ODBC* becoming C/C++ standard on Windows
  - Sun's *JDBC* a Java equivalent
  - Supposedly DBMS-neutral
    - a "driver" traps the calls and translates them into DBMS-specific code
    - database can be across a network

SQL API in Java (JDBC)

```
Connection con = // connect
   DriverManager.getConnection(url, "login", "pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT COF_NAME, PRICE FROM COFFEES"
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
    // loop through result tuples
    while (rs.next()) {
        String s = rs.getString("COF_NAME");
        Float n = rs.getFloat("PRICE");
        System.out.println(s + "   " + n);
    }
} catch(SQLException ex) {
    System.out.println(ex.getMessage ()
        + ex.getSQLState () + ex.getErrorCode ());
}
```


Concurrency Control & Recovery

Why Have Concurrent Processes?

- Better transaction throughput, response time
- Done via better utilization of resources:
  - While one processes is doing a disk read, another can be using the CPU or reading another disk.
- **DANGER DANGER!** Concurrency could lead to incorrectness!
  - Must carefully manage concurrent data access.
  - There's (much!) more here than the usual OS tricks!

Transactions

- Basic concurrency/recovery concept: a *transaction (Xact)*.
  - A sequence of many actions which are considered to be one atomic unit of work.
- DBMS "actions":
  - reads, writes
  - Special actions: commit, abort
  - for now, assume reads and writes are on tuples; we'll revisit this assumption later.

The ACID Properties

- A tomicity: All actions in the Xact happen, or none happen.
- C onsistency: If each Xact is consistent, and the DB starts consistent, it ends up consistent.
- I solation: Execution of one Xact is isolated from that of other Xacts.
- Durability: If a Xact commits, its effects persist.

Passing the ACID Test

- Concurrency Control
  - Guarantees Consistency and Isolation, given Atomicity.
- Logging and Recovery
  - Guarantees Atomicity and Durability.
- C. C.:
  - What problems could arise?
  - What is acceptable behavior?
  - How do we guarantee acceptable behavior?

Schedules

- Schedule: An interleaving of actions from a set of Xacts, where the actions of any 1 Xact are in the original order.
  - Represents some actual sequence of database actions.
  - Example: $R_1(A)$, $W_1(A)$, $R_2(B)$, $W_2(B)$, $R_1(C)$, $W_1(C)$
  - In a *complete* schedule, each Xact ends in commit or abort.
- Initial State + Schedule ® Final State

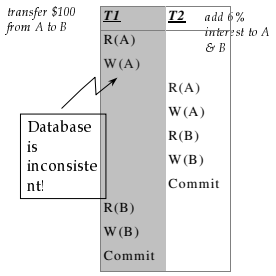
T1	*T2*
R(A)	
W(A)	
	R(B)
	W(B)
R(C)	
W(C)	

Acceptable Schedules

- One sensible "isolated, consistent" schedule:
  - Run Xacts one at a time, in a series.
  - This is called a serial schedule.
  - NOTE: Different serial schedules can have different final states; all are "OK" -- DBMS makes no guarantees about the order in which concurrently submitted Xacts are executed.
- Serializable schedules:
  - Final state is what *some* serial schedule would have produced.
  - Aborted Xacts are not part of schedule; ignore them for now (they are made to 'disappear' by using logging).

Serializability Violations

- Two actions conflict when 2 xacts access the same item:
  - W-R conflict: T2 reads something T1 wrote.
  - R-W and W-W conflicts: Similar.
- WR conflict (dirty read):
  - Result is not equal to any serial execution!



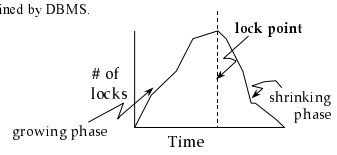
Locking: A Technique for C. C.

- Concurrency control usually done via locking.
- Lock info maintained by a "lock manager":
  - Stores (XID, RID, Mode) triples.
    - This is a simplistic view; suffices for now.
  - Mode Î {S,X}
  - Lock compatibility table:
- If a Xact can't get a lock, it is suspended on a wait queue.

	--	S	X
--	√	√	√
S	√	√	
X	√		

Two-Phase Locking (2PL)

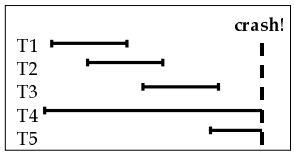
- 2PL:
  - If T wants to read an object, first obtains an S lock.
  - If T wants to modify an object, first obtains X lock.
  - If T releases any lock, it can acquire no new locks!
- Locks are automatically obtained by DBMS.
- *Guarantees* serializability!
  - Why?



Logging and Recovery: Motivation

- Atomicity:
  - Transactions may abort ("Rollback").
- Durability:
  - What if DBMS stops running? (Causes?)

❖ Desired Behavior after system restarts:
- T1, T2 & T3 should be durable.
- T4 & T5 should be aborted (effects not seen).



Assumptions

- Concurrency control is in effect.
  - Strict 2PL, in particular.
- Updates are happening "in place".
  - i.e. data is overwritten on (deleted from) the disk.
- A simple scheme to guarantee Atomicity & Durability?

Handling the Buffer Pool

- Force write to disk at commit?
  - Poor response time.
  - But provides durability.
- Steal buffer-pool frames from uncommited Xacts?
  - If not, poor throughput.
  - If so, how can we ensure atomicity?

	No Steal	Steal
Force	Trivial	
No Force		Desired

Basic Idea: Logging

- Record REDO and UNDO information, for every update, in a *log*.
  - Sequential writes to log (put it on a separate disk).
  - Minimal info (diff) written to log, so multiple updates fit in a single log page.
- Log: An ordered list of REDO/UNDO actions
  - Log record contains:
    <XID, pageID, offset, length, old data, new data>
  - and additional control info

Παγκόσμιος Ιστός και Βάσεις Δεδομένων

Ο Παγκόσμιος Ιστός

Κάθε αρχείο αναγνωρίζεται από μια μοναδική διεύθυνση (**universal resource locator**) **URL**:

```
http://www.uoi.gr/~pitoura/webdb00.html
```

Τα αρχεία είναι γραμμένα χρησιμοποιώντας την **Hypertext Markup Language** (**HTML**) και περιέχουν συνδέσεις (links) σε άλλα αρχεία

Ο Παγκόσμιος Ιστός

Web Browser (π.χ., Microsoft's Internet Explorer, Netscape Navigator) μεταφράζει και απεικονίζει τέτοια αρχεία

Web site συλλογή από τέτοια αρχεία

Web Server ένα πρόγραμμα που δέχεται τα URLs και επιστρέφει τα αντίστοιχα αρχεία

Ο **παγκόσμιος ιστός (World Wide Web** ή **Web)** είναι η συλλογή των web sites που είναι προσπελάσιμες μέσω του Διδικτύου.

Ο Παγκόσμιος Ιστός

Μια HTML σύνδεση (link) περιέχει μια URL

- Όταν ο χρήστης επιλέγει μια σύνδεση ο Web browser συνδέεται με το Web server του προορισμού μέσω ενός πρωτοκόλλου σύνδεσης που λέγεται HTTP και δίνει τη URL του συνδέσμου.

- Όταν ο Web browser λάβει το αρχείο από το Web server ελέγχει τον τύπο του αρχείου εξετάζοντας την κατάληξη του και το απεικονίζει αν χρειαστεί καλώντας το αντίστοιχο πρόγραμμα (π.χ., acroread ή Microsoft World)

Στατικές και δυναμικές σελίδες

Ο Παγκόσμιος Ιστός: URL

URL παρέχει ένα global naming scheme (http://www.w3.org/Addressing/URL)

(ίσως να μη δουλεύει - μέση διάρκεια ζωής 44 μέρες!)

Περιλαμβάνει το όνομα του πόρου αλλά και το πρωτόκολλο που θα χρησιμοποιηθεί για την προσπέλαση του. Παραδείγματα:

- ftp://server:port/resourcename
- jdbc:dbsystem://server:port/resourcename (όπου dbsytem είναι το όνομα ενός ΣΔΒΔ πχ Informix. JDBC είναι ένα interface για προσπέλαση σχεσιακών βάσεων δεδομένων από εφαρμογές Java)
- http://server:port/virtualdirectory/owa/packagename.storedprocedurename
- http://server:port/db2www/?langenv=dtw_sql(perform)&sql=''select+*+from+opera'' (database query URL)

Ο Παγκόσμιος Ιστός: HTML

HTML document markup language -- tags

```
<HTML>
<HEAD></HEAD>
<BODY>
science:
<ul>
   <li> Author: Richard Feynman</li>
   <li> Title: The Character of Physical Law </li>
   <li> Published: 1980</li>
</ul>
...
</BODY>
</HTML>
```


Ο Παγκόσμιος Ιστός: HTML

Audio - video αλλά και προγράμματα Java

Forms

αν η URL που ζητήθηκε περιέχει μια form, ο web server επιστρέφει τη form στον web browser, ο χρήστης συμπληρώνει τα πεδία, και η formεπιστρέφεται στο web server όπου οι τιμές των πεδίων μπορεί να χρησιμοποιηθούν ως είσοδοι σε προγράμματα που εκτελούνται στο ίδιο μηχάνημα με τον server

Hypertext links

Έχουν προταθεί επεκτάσεις της HTML με database constructs

Ο Παγκόσμιος Ιστός: HTTP

A client/server, content neutral communication protocol that runs over TCP/IP

Είδη αιτήσεων

• GET retrieve static or dynamic content - οι παράμετροι περνούν μέσω της αντίστοιχης URL

• POST retrieve dynamically generated resources- οι παράμετροι στέλνονται ξεχωριστά και όχι μέσω του URL - οι σελίδες που παράγονται συνήθως δε γίνονται cache

• PUT - DELETE για να προστεθούν, αντικατασταθούν ή σβηστούν πόροι στο server

Ο Παγκόσμιος Ιστός: HTTP

• STATELESS

• new connection for each inline image

Διαδικτυακές Βάσεις Δεδομένων

• **Great** for URL-based sharing of files(ftp ..), documents (html), services (e.g., forms: travel, auctions, ..) community resources (e.g., DB&LP site) global knowledge (sometimes!)

• **Stinks** for content-based queries

(data is buried: web pages, news groups, clasified ad sites, music store inventories

no schems

no data types)

διανασμένο από διαφάνειες του M. Carey για το μάθημα: "Post-Modern Database Systems: Databases Meet the Web", Berkeley 1998-99

Διαδικτυακές Βάσεις Δεδομένων

• Αποθήκευση και διαχείριση web πόρων σε σχεσιακές βάσεις δεδομένων

• μοντελοποίηση - ερωτήσεις - προσπέλαση στο web χρησιμοποιώντας τεχνικές σχεσιακών βάσεων δεδομένων

• προσπέλαση σχεσιακών βάσεων δεδομένων από το web

• Αποθήκευση σχεσιακών δεδομένων στο web

• κατασκευή και διαχείριση σελίδων