

CSCI0320

Introduction to Software Engineering

Lecture 11

Databases

Motivation

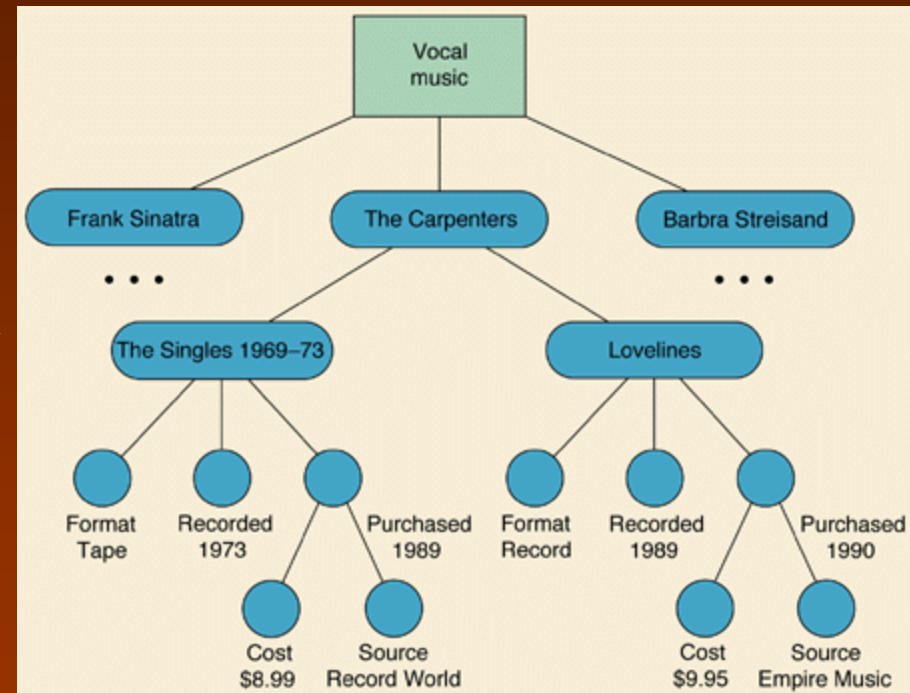
- **There is a lot of data available**
 - **People want to use it**
- **Freedb: database of all CDs published**
 - 4G of data (one big text file)
 - Semi-parsable text records
 - Provide disk title, artist, genre
 - Provide track title, artist, offsets
 - Provide additional information
- **Suppose you wanted to use this data**
 - **How would you find your favorite CD?**
 - **How would you determine how many CDs for a given song?**

Data Organization

- **Suppose you had lots of memory**
 - **How would you organize the data?**
- **Suppose you wanted to find**
 - All CDs of a given artist
 - All CDs with a given title
 - A particular song
 - All CDs containing a certain phrase in title
- **What if you don't know in advance what people are going to ask for**
 - How would you organize the data

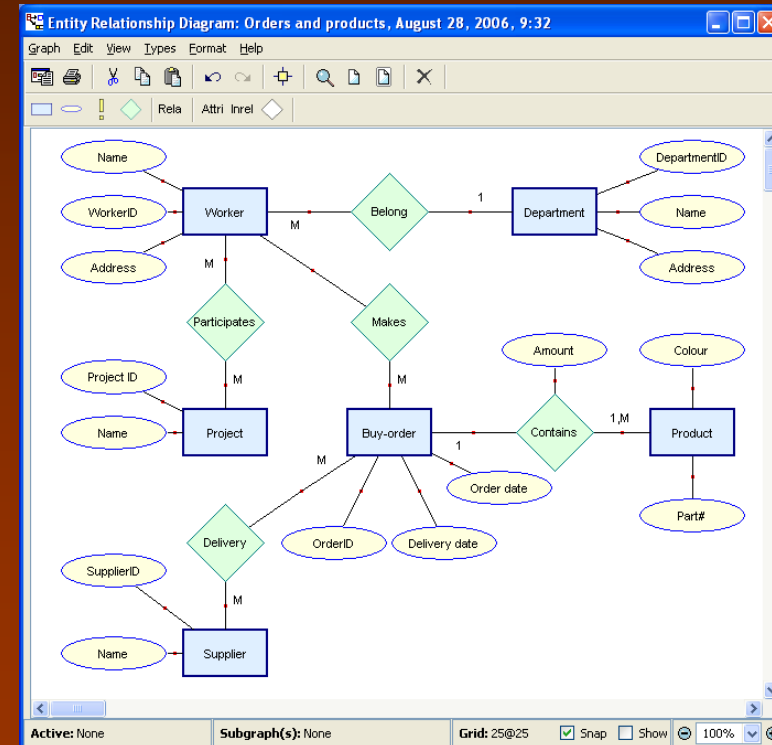
Hierarchical Databases

- Data organized as a **tree**
 - You started at root
 - Could then query elements under root
- Levels user settable
 - Can be indexed – fast access by key
 - Can be sequential
- Data storage on disk matched hierarchy



Network Databases

- Hierarchies are limited
 - Instead of trees, use **graphs**
- **Entities & Relationships**
 - Entities represent data basics
 - Relationships show how basics are related
- **Query by traversal**



Need for More

- **These were limiting**
 - Require **anticipating user's queries**
 - Since query implies navigation
 - **Want an organization**
 - That is independent of the queries
 - That can handle any query efficiently
 - At least potentially
- **Result was the relational database model**
 - Data organized into tables
 - Relationships between tables
 - Defined dynamically as part of query

Relations or Tables

- Basic unit for data organization
 - Table has **rows** and **columns**
 - **Columns** are typed **FIELDS**
 - **Rows** are **TUPLES** representing one data element

Person

Login	LastName	FirstName
skol	Kovalevskaya	Sofia
mlom	Lomonosov	Mikhail
dmitri	Mendeleev	Dmitri
ivan	Pavlov	Ivan

Project

ProjectId	ProjectName
1214	Antigravity
1709	Teleportation
1737	Time Travel

Experiment

ProjectId	ExperimentId	NumInvolved	ExperimentDate	Hours
1214	1	1	NULL	1.5
1214	2	1	1889-11-01	14.3
1709	1	3	1891-01-22	7.0
1709	2	1	1891-02-23	7.2
1737	1	1	1900-07-05	-1.0
1737	2	2	1900-07-05	-1.5

Involved

ProjectId	ExperimentId	InvolvedId	Login
1214	1	1	mlom
1214	2	1	mlom
1709	1	1	dmitri
1709	1	2	skol
1709	1	3	ivan
1709	2	1	mlom
1737	1	1	skol
1737	2	1	skol
1737	2	2	ivan

~f 2006

Table Concepts

- **Tables are stored sequentially on disk**
 - Tuple by tuple
- **You do queries over tables**
 - Try to find tuples that have given properties
- **Combine tuples from multiple tables**
 - Based on the data values
 - Build a new tuple from other tuples
- **Result is a new table**
 - Containing output tuples

Problems that Arise

- **How to make this efficient**
- **How to handle pointers or references**
- **How to avoid duplicate information**
 - **When updating the database**
- **How to avoid inconsistent information**
 - **When updating the database**

CD Database

Disk

ID	Title	ArtistID	Length	Genre	Year

Track

ID	Name	DiskId	ArtistId	Length	Number	Offset

Artist

ID	Name

CD Database

Extended

DiskID	TrackID	Data

Words

Word	Type	ID

Type: T => Title, A => Artist, N=>Track name,
D => Disk Data, I => Track data

Why This Format

- Why have artist as a separate relation?
- Why have ID field in Disk, Track?
- Why have Words as a separate relation?
- Why have length in disk relation?
- Why have length & number in track?

Simple Query

- **Find all CDs that have “alive” in the title**
 - Step through the Disk table
 - Check each title for “alive”
 - Output tuples that match
 - Find the word “alive” in Words table
 - With type = T
 - For each tuple that matches, use the ID to find
 - The corresponding element in the Disk table
 - Which is more efficient?
 - What is efficiency here?
 - Which could be made more efficient?

Sample Queries

- **Find all CDs that have “alive” and “Paris” in the title**
 - How would you implement this?
 - Could you use the Words table?
- **Find all CDs with “Beatles” as an artist**
 - How could you implement this?

Tables As Results

- **The result of a query**
 - Set of matching tuples
 - But this is a table
- **Result of a query is a table**
 - Can be used in another query
 - Can be treated as an original table

Indices

- **To achieve efficiency, use **indices****
 - **Fast access to tuples in a relation**
 - **Based on a **key****
 - Can be single field
 - Can be multiple fields
 - **Key does not have to be unique**
- **How would you implement an index?**
 - **Data structure on disk**
 - **Balanced tree with blocked records**
 - **Hash table**
- **Why not index everything?**

Indices for CD Database

- **Disk:** ID, ArtistID
- **Track:** ID, ArtistID, DiskID, Name
- **Extended:** DiskID, TrackID
- **Artist:** ID, Name
- **Words:** Word

Query Language: SQL

- **Want a language for defining tables**
 - Based on other tables
 - Using fields and values
- **History of such languages**
 - **Relational calculus** { x : expression }
 - Quel (Berkeley)
 - **Relational algebra**
 - Operations: select, project, product, join
 - **SEQUEL**: higher level language
 - **SQL**: cleaned up version of above
 - What is used today almost exclusively

SQL Select Basics

- **SELECT** is the query mechanism
 - **SELECT** <result>
 - **FROM** <source relations>
 - **WHERE** <condition>
- **FROM**
 - List of relations or <Relation ID> pairs
 - ID provides variable for easier access
 - Indicates where the data is coming from
 - Same relation can be listed multiple times

SQL Select Basics

- **SELECT**
 - Defines the resultant relation
 - List of fields and where they come from
 - <expression>
 - <expression> as <name>
 - * for everything
 - COUNT(*) and other group expressions

WHERE Clause

- **Sequence of relationals**
 - Separated by AND and OR
 - X.field = 'value'
 - X.field = Y.value
- Can also included nested **SELECT**
 - X.field IN (SELECT ...)
- Can also set operations on tables

Examples

- **Find all CDs with 'alive' in the title**
 - Simple form
 - **SELECT D.title FROM Disk D**
 - **WHERE D.title LIKE '%alive%'**
 - Using the Word table
 - **SELECT D.title**
 - **FROM Disk D, Words W**
 - **WHERE W.word = 'alive' AND**
 - **W.id = D.id AND W.type = 'T'**
- **psql -h bridget cdquery**

Sample Queries

- **Find all CDs with 'alive' & 'Paris' in title**
 - **SELECT D.title**
 - **FROM Disk D, Words W, Words W1**
 - **WHERE**
 - **W.word = 'alive' AND**
 - **W.id = D.id AND**
 - **W.type = 'T' AND**
 - **W1.word = 'Paris' AND**
 - **W1.id = D.id AND**
 - **W1.type = 'T'**

Sample Query

- **Find all CDs with 'Beatles' as an artist**
 - **SELECT D.title**
 - **FROM Disk d, Words w**
 - **WHERE w.word = 'beatle' and w.type='A'**
and d.artistid = w.id
- **Why no artist table?**
- **Can be written as a nested query**

Java and SQL

- Using psql is okay (actually **painful**)
 - How to use it from a program
 - Could run psql as a command
- Database runs as a separate process
 - Typically on a separate machine
 - **WHY?**
 - Handle multiple users simultaneously
 - Protect and isolate from user code
 - Communicate with the database
 - Using **sockets**
 - Message protocol

Message Protocol

- **Want to work at a higher level**
 - Send a query, read the results
 - Might not want all the results at once
 - Don't want to deal with messages
 - But you need standard messages
- **This has been done**
 - For multiple languages (same protocol)
 - For multiple database systems
 - **JDBC** for Java
 - **ODBC** for C++
 - **PHP** uses it with built-in data structures

Using JDBC

- **First need to load the library**
 - Each database system is different
 - **Why?**
 - **Class.forName(“postgresql.Driver”)**
- **Then you need to connect to the database**
 - **java.sql.Connection**
 - Represents the connection
 - A program can have multiple connections
 - But typically will only use one
 - **DriverManager.getConnection(nm,user,pw)**
 - nm = “jdbc:postgresql://bridget/cdquery
 - Catch SQLException
 - **sql_conn.close()** to disconnect

Issuing a Query

- Need to represent interaction with DBMS
 - **java.sql.Statement** does this
 - Can be a single query
 - Or a batch of updates
 - Or a transaction
 - Obtain using
 - **statement s = sql_conn.createStatement()**
 - Catch SQLException
 - Issue a query
 - **s.executeQuery(“SELECT ...”)**
 - Returns a **java.sql.ResultSet** object
 - Catch SQLException

Other Uses of Statement

- **To execute updates**
 - `s.executeUpdate(“sql command”)`
- **To send a set of commands at once**
 - `s.clearBatch()`
 - `s.addBatch(“...”);`
 - `s.executeBatch();`

Handling Query Results

- **java.sql.ResultSet**
 - Represents the result of a query
 - Acts as an **iterator** over tuples in the result
 - Lets you access fields of the current tuple
 - **Iterating**
 - `next()`: move to next (first) tuple
 - `first()`, `last()`, `previous()`, `absolute(int)`
 - **Accessing** fields
 - By index (1..n) or by name
 - Specified by type (automatic conversion)
 - `getString(int)`, `getString("field")`
 - `getInt`, `getDouble`, `getBytes`, `getTime`, `getDate`
 - `getTimestamp`
 - **Close** the result set using `close()`

XML Databases

- Suppose you wanted to query data
 - Where the data is in XML format
- What do queries look like?
 - Data relationships
 - Hierarchy relationships
- XPath expresses hierarchy relationships
 - Can specify arbitrary paths
 - With wildcards
- XQuery provides query language
 - Combines SQL like access to data
 - With XPath descriptions of paths

XML Implementation

- **How would you implement this?**
 - Internal representation of XML
 - Indices where appropriate
 - **What if the XML document is huge?**
- **Move everything to a relational database**
 - Database relations expression hierarchy
 - Map XPath to SQL expression
 - Add that to normal expression