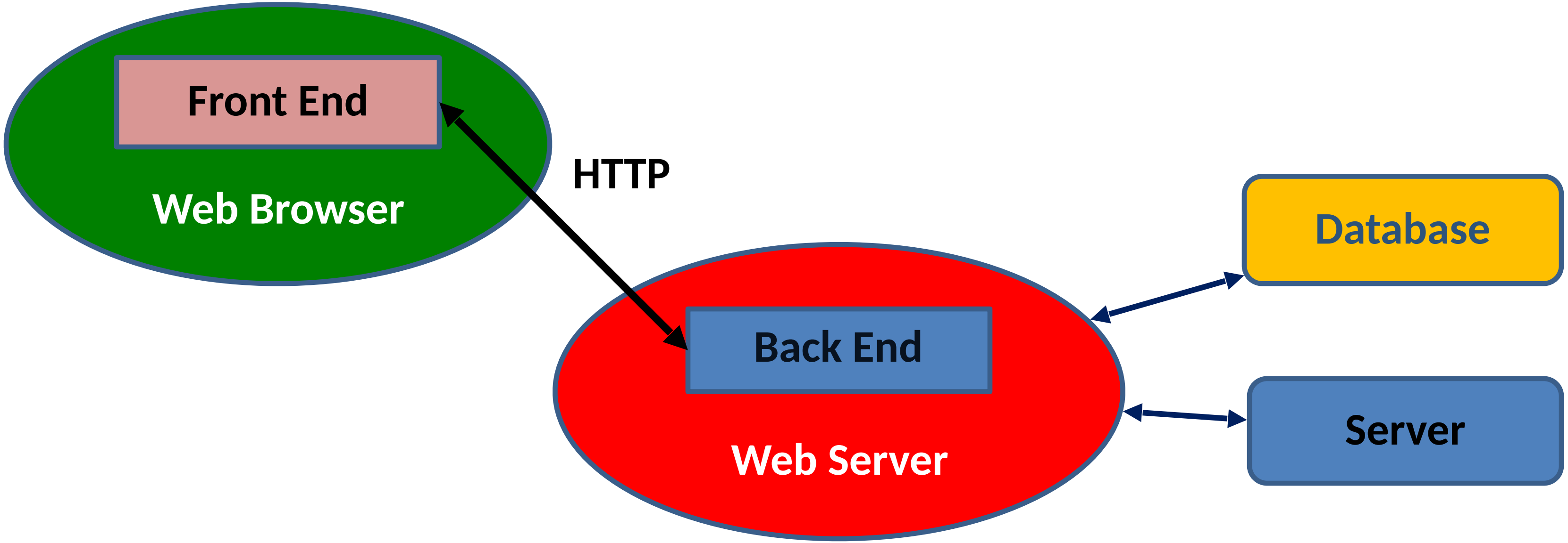
CS1320
Creating Modern Web and Mobile Applications

Lecture 18:

Databases I

Web Applications



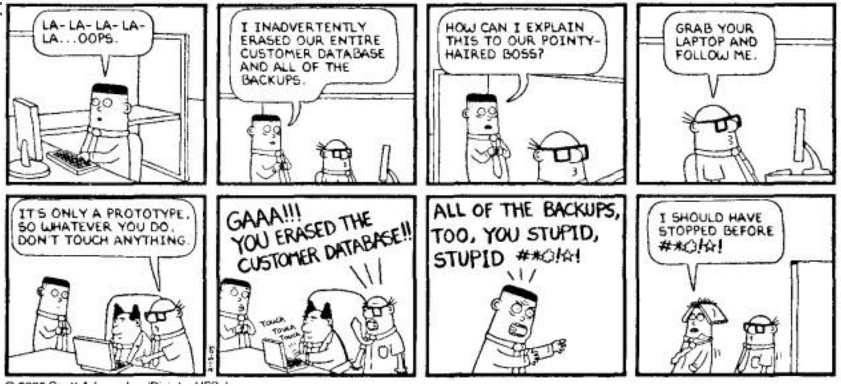
Why Use Databases

- Efficient ways of storing large amounts o
  - Designed to scale for data
  - Designed to scale for multiple servers
- Ensure the integrity of the data
  - Against hardware failures
  - Against multiple simultaneous updates
  - Against inconsistent updates
- Allow easy and efficient access to the data
  - Query language makes any access possible
  - Query optimization can make access efficient

Data Storage Requirements

Data demands	Usage	Key metrics	Best-fit storage device	Example
• Highest performance • Highest availability • Lowest latency • Lowest data footprint	Active data: Log, journal, paging, meta data or index files	Cost per IOPs Activity per watt	Enterprise SSD – Serial attached SCSI (SAS)	Pulsar 2.5-inch SSD
• High performance • High availability • Low latency • Low data footprint	Primarily active data: Email, database, video, virtual machine, virtual desktop, web server	Cost per IOPs Activity per watt	15,000 rpm or 10,000 rpm SAS	Savio 15K 2.5-inch HDD Savio 10K 2.5-inch HDD
• Good performance • High data footprint • Low power • Low cost per GB	Mix of active and inactive data, bulk storage: File server, data warehousing, mining, analytics, backup, disaster recovery	Cost per GB Cost per watt	7,200 rpm SAS/ Serial Advanced Technology Attachment (SATA)	Constellation 2.5-inch HDD Constellation CS 3.5-inch HDD Constellation ES 3.5-inch HDD
• Highest data footprint • Lowest power	Primarily inactive data, tape replacement: Archive, backup for long-term data retention	Cost per TB Cost per watt	Tape now, < 7,200 rpm, HDD possibly in the future	

- The storage needs to be robust
  - Don't want to lose a user's purchase
- Need to handle multiple requests at once
  - Only one person can buy a particular airplane seat
- The data is important
  - The data is the company



Securing the Data

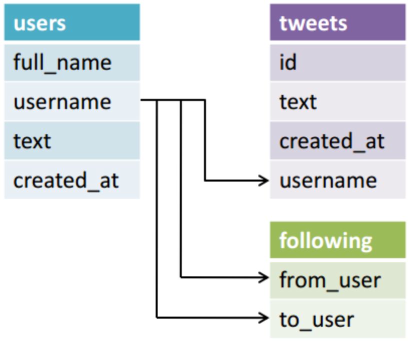
- Providing robust data storage is complex
  - But a common necessity
  - Can be separated from the actual application
  - Done by database systems so you don't have to
    - Database system is 20% code to do the work
    - And 80% code to handle exceptions, errors, conflicts, recovery
- Database systems are integral to web applications

Data Example: FreeDB

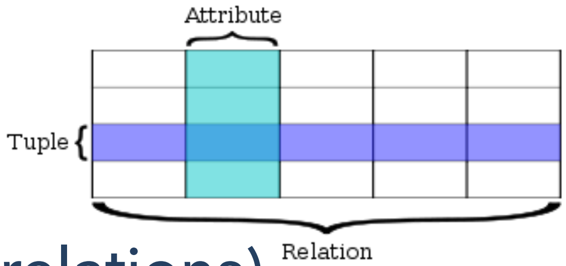
- Database of all CDs published
- Provided as lots of text files in a tar image
  - With semi-parsable records (not XML or JSON)
  - Provide disk title, artist, genre, length
  - Provide track title, artist, offsets
  - Provide additional information and comments
- It would be nice to have web access to this data
  - To find CDs
  - To add comments
  - To correct mistakes and typos (lots of these)
  - To manage ones own collection
- We've considered the application earlier
  - Now lets consider the data

Relational Databases (SQL)

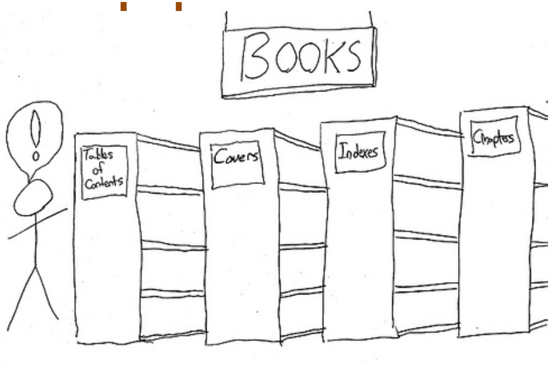
- Organize the data in a way that is (somewhat) independent of its use
  - Can support arbitrary queries of the data
  - Don't have to know what the queries are in advance
- In general data consists of
  - Facts (actual data)
  - Relationships between the facts (pointers)
- Relational databases make all relationships im|
  - Based on matching values, not on links (pointers)



Relational Databases



- A relational database is a set of TABLES (relations)
  - Each table holds a coherent set of data
- A table is divided into FIELDS (columns, attributes)
  - Each field holds data of a single (simple) data type
- The table's ROWS (tuples) contain the a
  - Value for each field of the table
  - A row is a single data instance
- One (or more) fields might be the KEY
  - Uniquely identify the row



If libraries were like relational databases

Relational Data

Hypothetical Relational Database Model

PubID	Publisher	PubAddress
03-4472822	Random House	123 4th Stree, New York
04-7733903	Wiley and Sons	45 Lincoln Blvd, Chicago
03-4859223	O'Reilly Press	77 Boston Ave, Cambridge
03-3920886	City Lights Books	99 Market, San Francisco

AuthorID	AuthorName	AuthorBDay
345-28-2938	Haile Selassie	14-Aug-92
392-48-9965	Joe Blow	14-Mar-15
454-22-4012	Sally Hemmings	12-Sep-70
663-59-1254	Hannah Arendt	12-Mar-06

ISBN	AuthorID	PubID	Date	Title
1-34532-482-1	345-28-2938	03-4472822	1990	Cold Fusion for Dummies
1-38482-995-1	392-48-9965	04-7733903	1985	Macrame and Straw Tying
2-35921-499-4	454-22-4012	03-4859223	1852	Fluid Dynamics of Aquaducts
1-38278-293-4	663-59-1254	03-3920886	1967	Beads, Baskets & Revolution

CDQuery Primary Database Schema

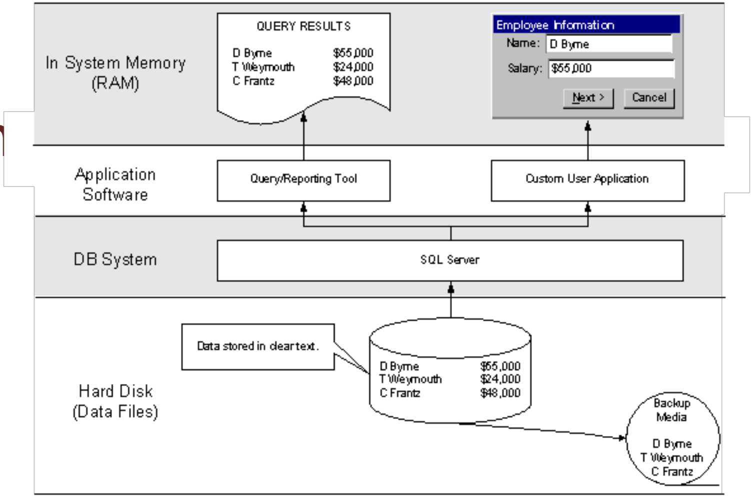
artist	ID	NAME

disk	ID	Title	ArtistID	Length	Genre	Year

track	ID	Name	DiskID	ArtistID	Length	Number	Offset

Obtaining Information from the Database

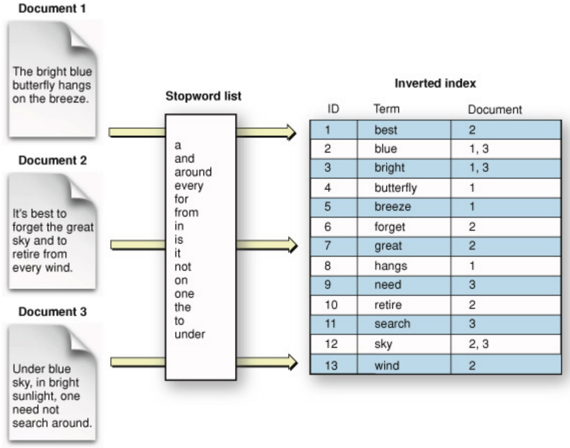
- Want to get information from the database
  - Not all of it at once
  - Information for a particular purpose
  - To answer a particular question
- What might we want to get from th
  - What would you like to know?



Sample Questions to Ask

- CDs of a given artist
- CDs with a particular song
- What artist is on the most CDs
- CDs are associated with 'nsync'
- What CDs have a track by Taylor Swift
- CDs that have 'Paris' in the title with artist Jacques Brel

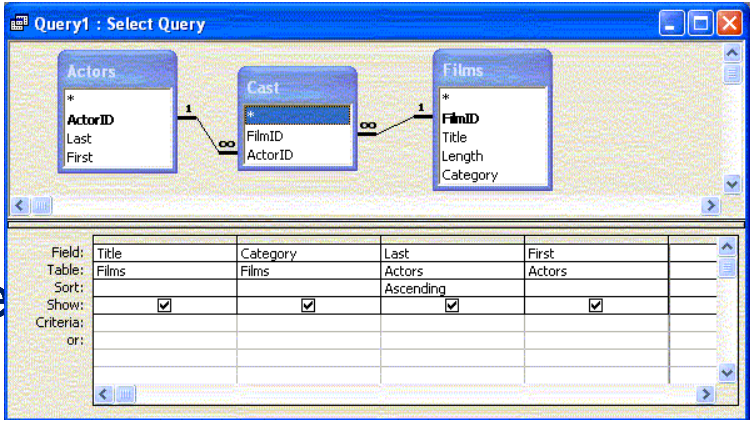
Inverted Indices



- Create a new relation to simplify text lookup
- Word and where it occurs
  - This is an inverted index
- Wherever a word occurs
  - Type: T=title, N=track name, A=artist, D=disk data, I=track data
  - ID: disk id, artist id or track id
  - Alternatives: multiple word relations, word number

words	Word	Type	ID

Example Query



CDs that have the word 'Paris' in the

SELECT d.title

FROM disk d, words w

WHERE w.word = 'paris' AND w.type = 'T' AND w.id = d.id

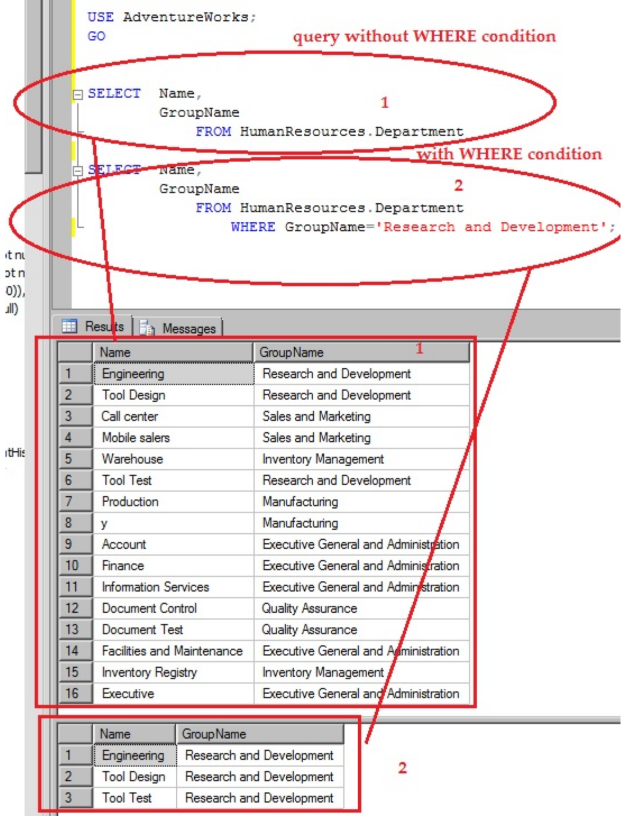
SQL Basics

- SELECT <result> FROM <source> WHERE <condition> [ORDER BY <col> ASC]
- SELECT: define the resultant table (result is a database table
  - List of fields and where they come from
  - Expression, Expression AS name, *
    - Generally just ***table.field_name***
    - Can be real expressions: ***field + 1***
    - Can be grouping expressions: ***COUNT(*)***
  - Can also specify ***DISTINCT***
- FROM: what tables to use as the input
  - Either a list of table names or <***table_name variable***> pairs
  - Variables provide shorted names for easier access
  - Variables allow tables to be listed multiple times
- Note case is sometimes important (mysql table names)

```
SELECT   sect.section_course_term_code  as term_code
         ,sect.course_reference_number  as CRN
         ,instr.instuctor_full_name as Instructor
         ,ques.question_text  as Question
         ,AVG(det.scaled_response_number)  as Overall_Rating
FROM    course_eval_responses_detail  det
        ,course_eval_section_dimension sect
        ,course_eval_instructor_dim instr
        ,course_eval_question_dimension ques
WHERE  det.course_eval_section_key  = sect.course_eval_section_key
AND      det.course_eval_instructor_key  =
instr.course_eval_instructor_key
AND      det.course_eval_question_key  =
ques.course_question_instructor_key
AND      ques.question_id = '0010'
AND      sect.section_course_term_code  = '201301'
GROUP BY  sect.section_course_term_code
             ,sect.course_reference_number
             ,ques.question_text
ORDER BY  sect.section_course_term_code
             ,sect.course_reference_number  DESC
```

SQL WHERE

- WHERE clause specifies the actual query
- Sequence of relational expressions
  - Separated by **AND** and **OR**
  - X.field = 'value'
  - X.field = Y.field
- Can also have nested SELECTS
  - X.field IN (SELECT …)
- Also set operations on tables
- Also string operations
  - Value **LIKE** pattern
    - % is a wildcard, _ matches any single character
  - Some database systems allow regular expressions (not standard)



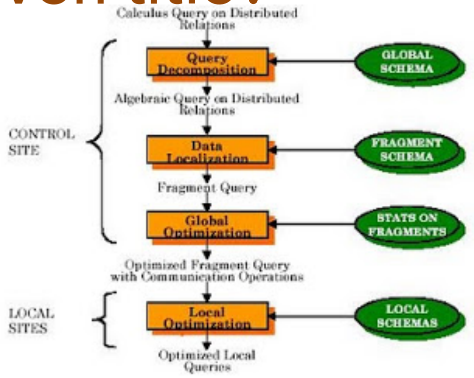
SQL Examples

- SELECT d.title FROM disk d
  WHERE d.title LIKE '%Paris%'
- SELECT d.title FROM disk d, words w
  WHERE w.word = 'paris' AND w.type = 'T' AND w.id = d.id
- SELECT d.title FROM disk d, words w
  WHERE w.word = 'beatles' AND w.type = 'A' AND d.artistid = w.id
- SELECT count(d.title) FROM disk d, artist a
  WHERE a.id = d.artistid AND a.name = 'Madonna'
- SELECT DISTINCT d.title FROM disk d, track t, words w
  WHERE w.word = 'madonna' AND w.type = 'A'
  AND t.artistid = w.id AND t.diskid = d.id

PostgreSQL
the world's most advanced open source database

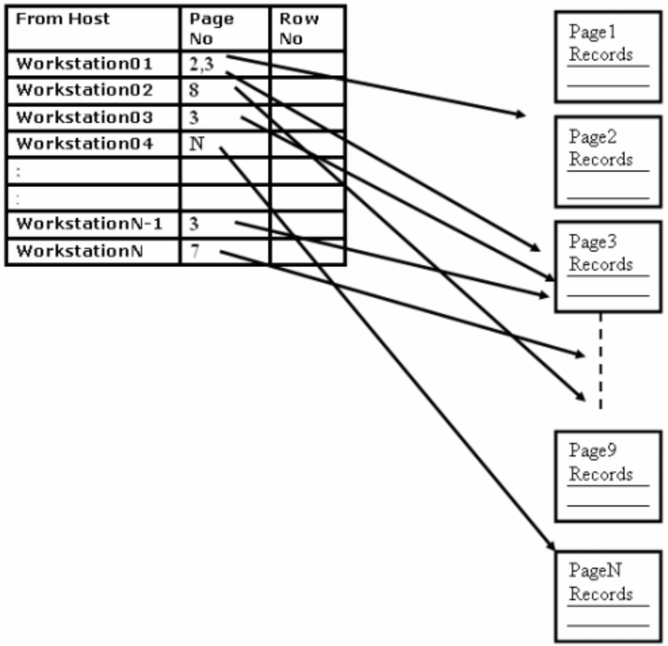
Queries

- Assume the data is stored on disk as tables
  - Table contains rows
  - Rows contain data
- How might the database find CDs with a given title?
  - Scanning a table to look for entries
  - Creating an index for the table
    - Fast access based on a value



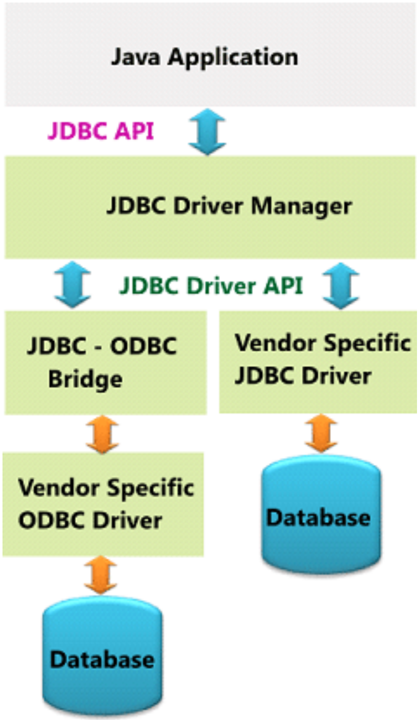
Indices

- How do indices work
  - How do they work in memory
    - Hash tables, trees (red-black trees)
    - Looking for a range of values rather than a particular value
  - B-trees (block-trees) versus Binary trees
    - Balanced trees with variables number of keys at each level
    - Minimize I/O operations
    - Can be scanned or accessed as a index
  - Bucket-based hash tables
- Indices can cover multiple columns at once
- Why not index everything?
  - How many indices are needed for 10 fields?
  - What is the cost of an index?
    - Storage, update time, creation time
  - Actually, this is one of today's trends – column-store databases



Embedded SQL

- SQL is used inside programs
  - Actually built into some languages
  - Create a string representing the query
  - Pass that string to the database to interpret
- Concepts
  - **Connection**: connection to the database
    - Accessed by a URL giving type, host, database, user, pwd,…
  - **Statement**: set of SQL statements to be executed as one
    - Typically a single query or set of updates
  - **ResultSet**: iterator over a returned table
    - Can iterate over tuples in the returned values
    - Can access fields of the current tuple
    - Note that the results are returned incrementally

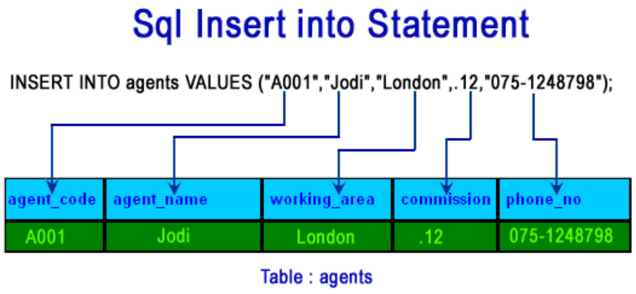


Using Embedded SQL Safely

- Queries and updates are built as strings
  - Provides flexibility
  - Lets program add user-entered values to the queries
  - Can be problematic if user values are non-standard
- Prepared statements
  - Queries with variables are defined as strings
  - Variables in the query are represented by $, $i, or ?
  - Values are passed when the query is used
  - Can be faster (database can optimize the query once)
  - Safer and more secure
  - ***Use when possible (and its always possible)***

```
import cx_Oracle

con = cx_Oracle.connect('pythonhol/welcome@localhost/orcl')

cur = con.cursor()
cur.prepare('select * from departments where department_id = :id')

cur.execute(None, {'id': 210})
res = cur.fetchall()
print res

cur.execute(None, {'id': 110})
res = cur.fetchall()
print res

cur.close()
con.close()
[pythonhol@localhost ~]$
```

SQL INSERT Statement

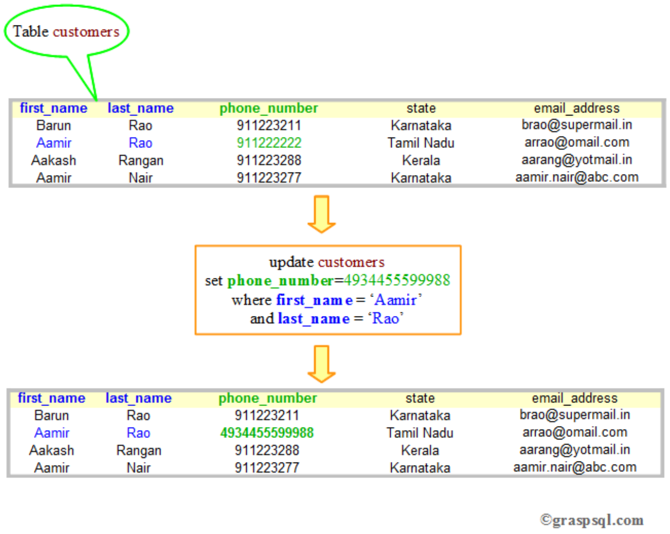


- INSERT INTO table (field, …, field)

  VALUES (val,…val)
  - List of fields is optional, but should be there
    - Avoids problems in future, reminds reader of what is what
  - Values can be
    - DEFAULT
    - Constants [true, false, 0, …]
    - Built-in Functions [CURRENT_TIMESTAMP]
    - Variables – use $, $i, ? and pass values separately
    - Results of queries [(SELECT id FROM Artist A WHERE A.name = 'nsync')]
- INSERT INTO table (field, …, field) SELECT …

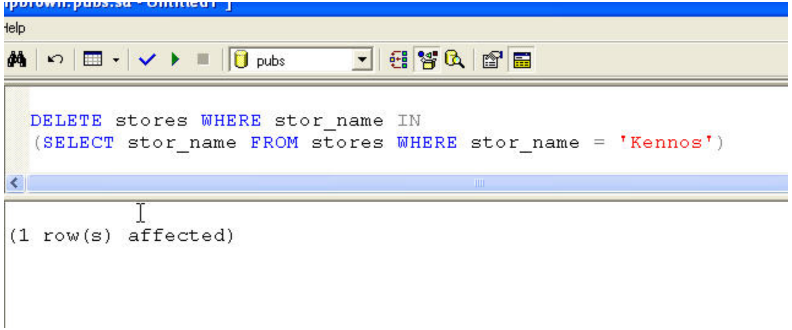
SQL UPDATE Statement

- UPDATE table SET field = value WHERE condition
  - SET field = value, field = value, …
  - Values as in an insert statement
  - WHERE as in normal select
- Can update multiple rows at once
  - Be careful with the condition



SQL DELETE Statement

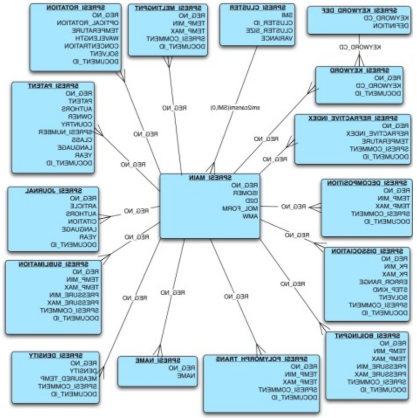
- DELETE FROM table WHERE condition
  - Removes all rows in table where condition is true
  - If condition is omitted, deletes all rows



Next Time

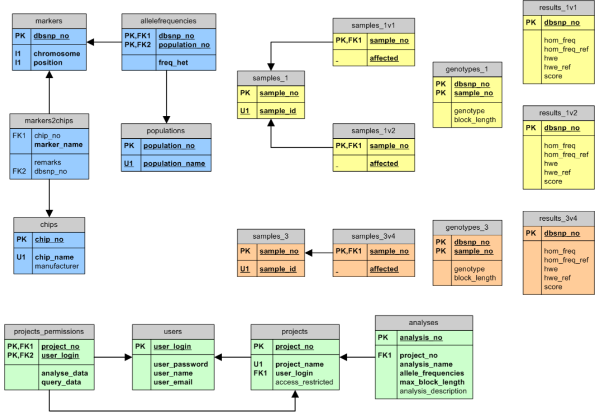
- More on databases

Data Organization



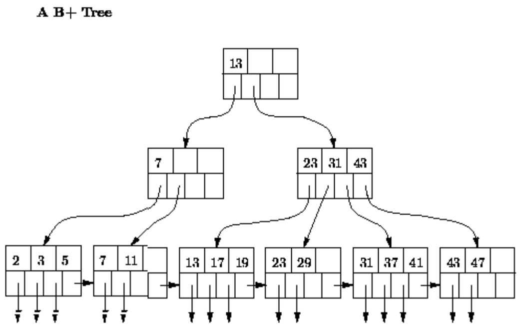
- Databases need to be efficient
  - The key to this is data organization
  - Structuring and organizing the database for performance
- How would you organize the CD data in a program?
  - Might think about objects – what are the objects?
  - Might think about access to data (e.g. Java Maps)
  - Why is this a trick question?

Data Organization



- Suppose you want to find
  - All CDs with a given title
  - All CDs with a given artist
  - All CDs containing a particular song
  - All CDs containing a certain phrase in the title
  - All CDs containing a particular song by a particular artist
- Suppose you don’t know what you will be asked?

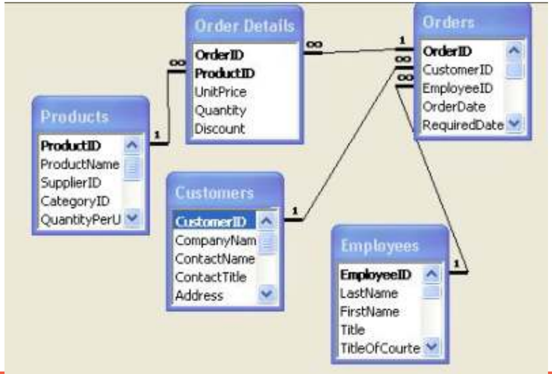
Disk Data Organization



- What happens when the data is on disk
  - Do the same algorithms and data structures apply?
  - How do you measure the costs in this case?
  - I/O operations rather than lookups, compares
    - What happens if you use solid-state disks?

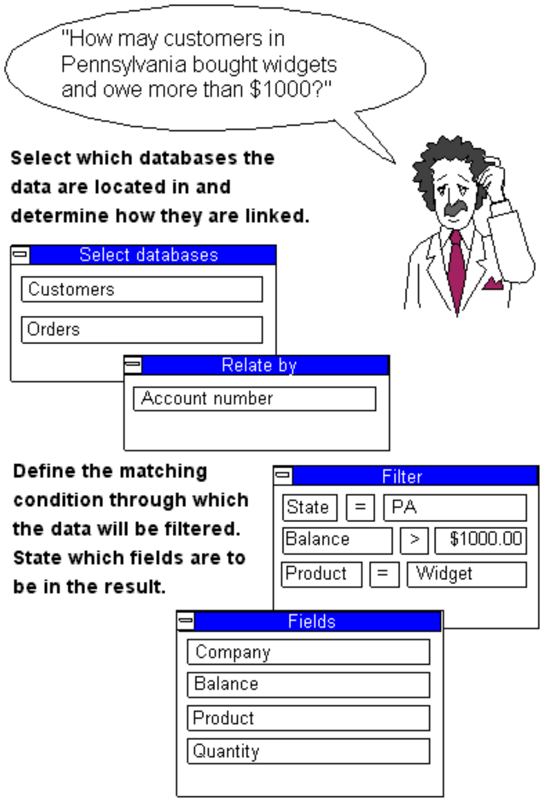
Relational Data Organization

- Traditional, in-memory, data organization
  - Data is represented explicitly
  - Relationships are represented explicitly (as links)
  - This assumes you know what will be asked
- Relational data organization
  - Data is organized into tables (or relations)
  - Relationships between tables are IMPLICIT
    - Based on data values, not links or pointers
    - Defined dynamically as needed



Querying the Database

- We need a language to express what data we
  - Used to describe how to get the data
  - **Query language**
- What should the result of a query be?
  - Set of values
  - Set of related values
  - A table (just like the data tables in the database)
- Using tables as the result of queries
  - Is a nice clean model
  - Allows queries to be nested
  - Allows queries to define new tables
    - Both real and virtual



Query Languages

- We want a standard language to express q
  - Several languages have been developed
    - Find all X satisfying Y
    - Operations on tables: project, product, select, union, …
    - Query-by-example
  - All have equivalent power
    - Can do a lot, can't do everything (transitive closure)
- Language should also handle
  - Setting up the database (defining tables)
  - Changing values in the database (update)
  - Adding data to the database (insert)

query

Transcript	*StudId*	*CrsCode*	*Semester*	*Grade*
	_5678	_CS532	_S2002	

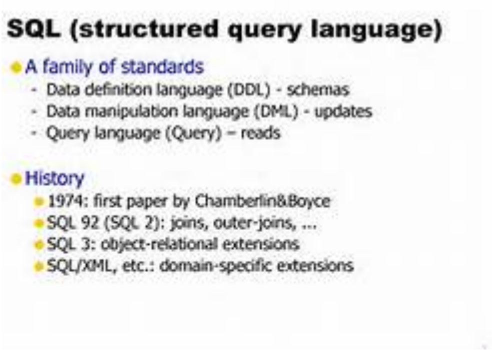
Teaching	*ProfId*	*CrsCode*	*Semester*
	_12345	_CS532	_S2002

update

HasTaught	*Professor*	*Student*
I.	_12345	_5678

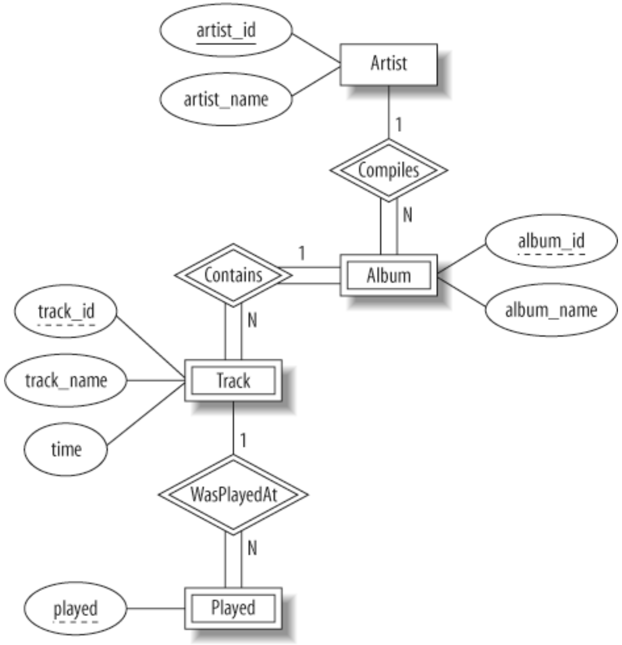
HasTaught	*Professor*	*Student*
P.		

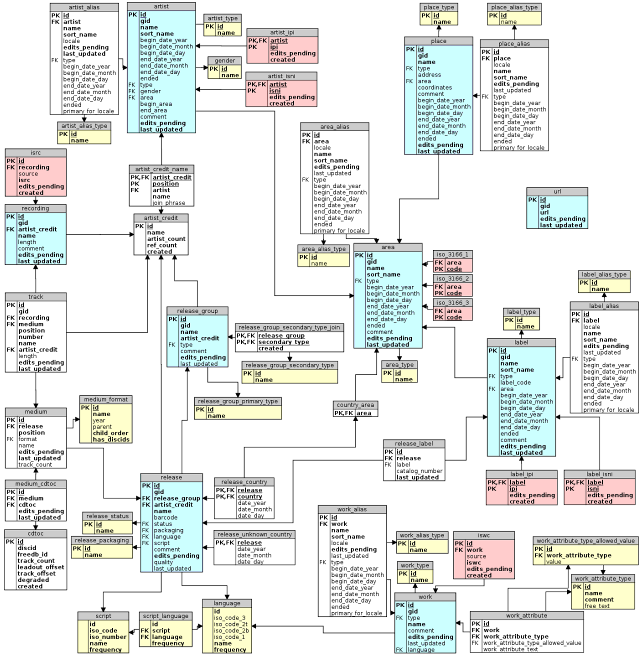
Query Languages



- SQL has become the standard
  - Language for building tables given tables
  - Used both directly and inside programs
  - SELECT for query
  - INSERT for insert
  - UPDATE for update
  - CREATE, GRANT, DROP, … for maintenance
- XQUERY is an extension to handle XML structures
- NOSQL is becoming more common for web apps

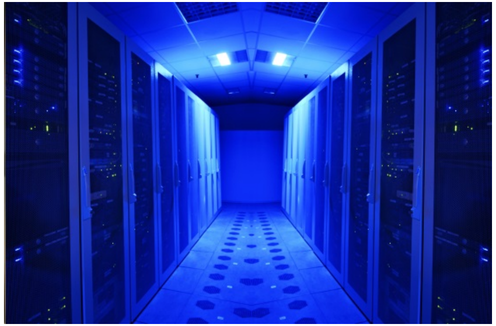
Entity-Relationship Diagrams





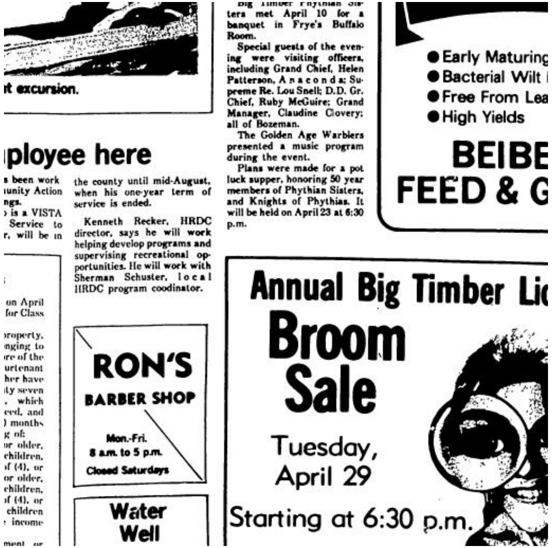
Data Storage

- Web applications are mainly data-centric
  - Generally in the form of web requests and web pages
  - Examples: amazon, expedia, …
- How do these applications work?
  - What data do they use?
  - What do they do with that data?
  - How do user interactions affect the data?
  - How does the data affect user interactions?



SwiftFeed Data

- What data is there
- What are the facts and relationships
- How could it be organized



Question

Which is not true about SQL databases?

A. They are designed to store large amounts of data
B. They provide robustness by safeguarding the data against hardware failure
C. They offer a generic interface for accessing the data
D. They let multiple users access and set data simultaneously
E. They make changing the data schema (format) easy

What the Database System Does

- Stores the data
- Ensures the integrity of the data
- Understands the query lan
- Compile & execute queries
- Allows data to be updated