Non-classical DBMSs

Database Trends (overview)

Future of Database

2/95

Core "database" goals:

- deal with very large amounts of data (petabyes, exabytes, ...)
- very-high-level languages (deal with data in uniform ways)
- fast query execution (evaluation too slow ⇒ useless)

At the moment (and for the last 30 years) RDBMSs dominate ...

- simple/clean data model, backed up by theory
- high-level language for accessing data
- 40 years development work on RDBMS engine technology

RDBMSs work well in domains with uniform, structured data.

... Future of Database

3/95

Limitations/pitfalls of classical RDBMSs:

- NULL is ambiguous: unknown, not applicable, not supplied
- "limited" support for constraints/integrity and rules
- no support for uncertainty (data represents *the* state-of-the-world)
- data model too simple (e.g. no direct support for complex objects)
- query model too rigid (e.g. no approximate matching)
- continually changing data sources not well-handled
- data must be "molded" to fit a single rigid schema
- database systems must be manually "tuned"
- do not scale well to some data sets (e.g. Google, Telco's)

... Future of Database

4/95

How to overcome (some) RDBMS limitations?

Extend the relational model ...

- add new data types and query ops for new applications
- deal with uncertainty/inaccuracy/approximation in data

Replace the relational model ...

- object-oriented DBMS ... OO programming with persistent objects
- XML DBMS ... all data stored as XML documents, new query model
- noSQL data stores (e.g. *(key,value)* pairs, json or rdf)

... Future of Database

5/95

How to overcome (some) RDBMS limitations?

Performance ...

- new query algorithms/data-structures for new types of queries
- parallel processing
- DBMSs that "tune" themselves

Scalability ...

- distribute data across (more and more) nodes
- techniques for handling streams of incoming data

... Future of Database

6/95

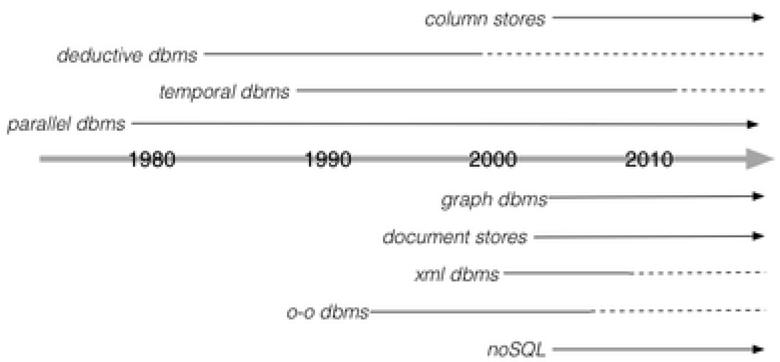
An overview of the possibilities:

- *"classical" RDBMS* (e.g. PostgreSQL, Oracle, SQLite)
- *parallel DBMS* (e.g. XPRS)
- *distributed DBMS* (e.g. Cohera)
- *deductive databases* (e.g. Datalog)
- *temporal databases* (e.g. MariaDB)
- *column stores* (e.g. Vertica, Druid)
- *object-oriented DBMS* (e.g. ObjectStore)
- *key-value stores* (e.g. Redis, DynamoDB)
- *wide column stores* (e.g. Cassandra, Scylla, HBase)
- *graph databases* (e.g. Neo4J, Datastax)
- *document stores* (e.g. MongoDB, Couchbase)
- *search engines* (e.g. Google, Solr)

... Future of Database

7/95

Historical perspective



Large Data

8/95

Some modern applications have massive data sets (e.g. Google)

- far too large to store on a single machine/RDBMS
- query demands far too high even if could store in DBMS

Approach to dealing with such data

- distribute data over large collection of nodes (also, redundancy)
- provide computational mechanisms for distributing computation

Often this data does not need full relational selection

- represent data via *(key,value)* pairs
- unique *key*s can be used for addressing data
- *values* can be large objects (e.g. web pages, images, ...)

... Large Data

9/95

Popular computational approach to such data: *map/reduce*

- suitable for widely-distributed, very-large data
- allows parallel computation on such data to be easily specified
- distribute (map) parts of computation across network
- compute in parallel (possibly with further *map*ping)
- merge (reduce) multiple results for delivery to requestor

Some large data proponents see no future need for SQL/relational ...

- depends on application (e.g. hard integrity vs eventual consistency)

Humour: Parody of noSQL fans (strong language warning)

Information Retrieval

10/95

DBMSs generally do precise matching (although `like`/regexps)

Information retrieval systems do approximate matching.

E.g. documents containing a set of keywords (Google, etc.)

Also introduces notion of "quality" of matching
(e.g. tuple T_1 is a *better* match than tuple T_2)

Quality also implies *ranking* of results.

Ongoing research in incorporating IR ideas into DBMS context.

Goal: support database exploration better.

Multimedia Data

11/95

Data which does not fit the "tabular model":

- image, video, music, text, ... (and combinations of these)

Research problems:

- how to specify queries on such data? ($image_1 \cong image_2$)
- how to "display" results? (synchronize components)

Solutions to the first problem typically:

- extend notions of "matching"/indexes for querying
- require sophisticated methods for capturing data features

Sample query: find other songs *like* this one?

Uncertainty

12/95

Multimedia/IR introduces approximate matching.

In some contexts, we have approximate/uncertain data.

E.g. witness statements in a crime-fighting database

"I think the getaway car was red ... or maybe orange ..."

"I am 75% sure that John carried out the crime"

Work by Jennifer Widom at Stanford on the *Trio* system

- extends the relational model (ULDB)
- extends the query language (TriQL)

Stream Data Management Systems

13/95

Makes one addition to the relational model

- *stream* = infinite sequence of tuples, arriving one-at-a-time

Applications: news feeds, telecomms, monitoring web usage, ...

RDBMSs: run a variety of queries on (relatively) fixed data

StreamDBs: run fixed queries on changing data (stream)

One approach: *window* = "relation" formed from a stream via a rule

E.g. StreamSQL

```
select avg(price)
from examplestream [size 10 advance 1 tuples]
```

Graph Data

14/95

Uses *graphs* rather than tables as basic data structure tool.

Applications: social networks, ecommerce purchases, interests, ...

Many real-world problems are modelled naturally by graphs

- can be represented in RDBMSs, but not processed efficiently
- e.g. recursive queries on `Nodes`, `Properties`, `Edges` tables

Graph data models: flexible, "schema-free", inter-linked

Typical modeling formalisms: XML, JSON, RDF

More details later ...

Dispersed Databases

15/95

Characteristics of dispersed databases:

- very large numbers of small processing nodes
- data is distributed/shared among nodes

Applications: environmental monitoring devices, "intelligent dust", ...

Research issues:

- query/search strategies (how to organise query processing)
- distribution of data (trade-off between centralised and diffused)

Less extreme versions of this already exist:

- grid and cloud computing
- database management for mobile devices

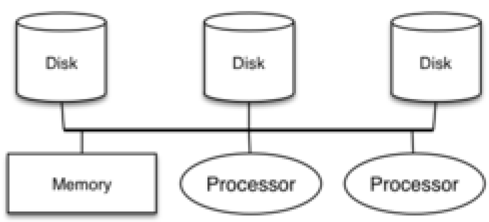
Parallel and Distributed Databases

Parallel and Distributed Systems

17/95

RDBMS discussion so far has revolved around systems

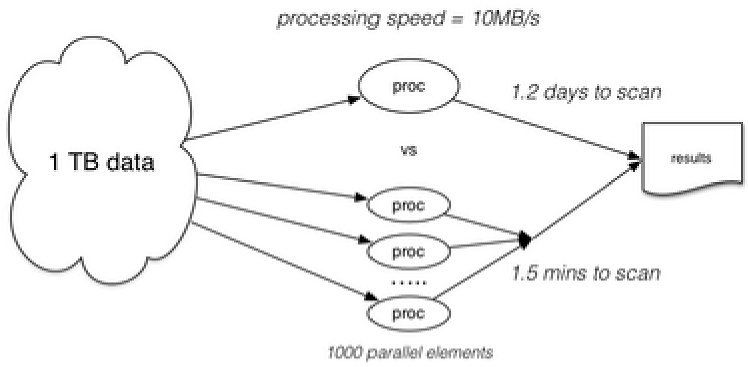
- with a single or small number of processors
- accessing a single memory space
- getting data from one or more disk devices



... Parallel and Distributed Systems

18/95

Why parallelism? ... Throughput!

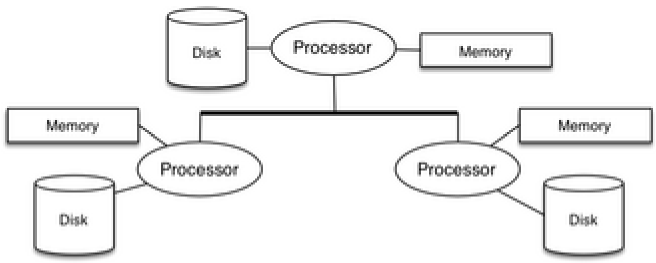


Parallel Architectures

19/95

Types: *shared memory*, *shared disk*, *shared nothing*

Example shared-nothing architecture:

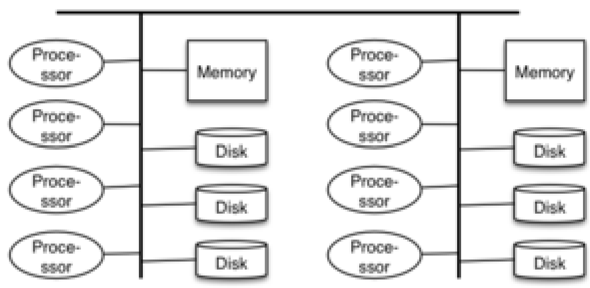


Typically in the same room (data transfer cost ~ 100's of µsecs)

... Parallel Architectures

20/95

Hierarchical architectures are hybrid parallel ones



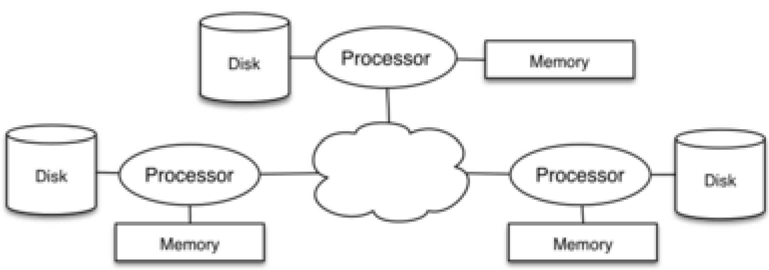
Typically on a local-area network (data transfer cost ~ msecs)

Distributed Architectures

21/95

Distributed architectures are ...

- effectively shared-nothing, on a global-scale network



Typically on the Internet (data transfer cost ~ secs)

Parallel Databases (PDBs)

22/95

Parallel databases provide various forms of parallelism ...

- processor parallelism can assist in speeding up memory ops
- processor parallelism introduces cache coherence issues
- disk parallelism can assist in overcoming latency
- disk parallelism can be used to improve fault-tolerance (RAID)
- one limiting factor is congestion on communication bus

PDBs typically run on closely-connected parallel architectures,
so we focus on hybrid architectures on a LAN.

Data Storage in PDBs

23/95

Consider each table as a collection of pages ...

Page addressing: *(Table, File, PageNum)*

- *Table* maps to a set of files (e.g. named by tableID)
- *File* distinguishes primary/overflow files
- *PageNum* maps to an offset in a specific file

If all data for one table resides on one node

- the above addressing scheme is adequate
- *Table* can identify *(Node, FileSet)*

... Data Storage in PDBs

24/95

However, with multiple nodes, we could ...

- *replicate* tables across several node
  - in which case, *Table* yields *{ (Node, FileSet) }*
- *partition* pages for one table across several nodes
  - in which case page addressing changes to include node
  - *(Node, Table, File, PageNum)*

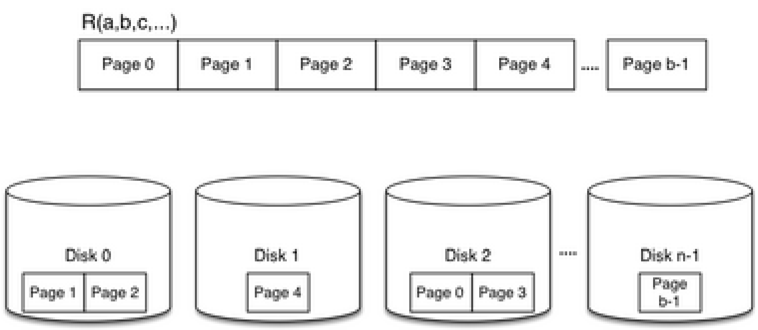
Could also have a combination of partitioning and replication

---

... Data Storage in PDBs

25/95

Data-partitioning example:



... Data Storage in PDBs

26/95

Data-partitioning strategies for one table:

- *round-robin* partitioning
  - cycle through nodes, each new tuple is added on the "next" node
- *hash* partitioning

    - use hash value to determine which processor and page
- *range* partitioning
    - ranges of attr values are assigned to processors

Assume: $R(a,b,c,...)$, D_0 .. D_{n-1} disks, tup_0 .. tup_{r-1} tuples

Storing data on many disks maximises chance for parallel data access

---

... Data Storage in PDBs 27/95

Round-robin partitioning:

- tuple t_i sent to D_j, tuple t_{i+1} sent to $D_{(j+1)\%n}$
- advantage: spreads data uniformly across disks
- disadvantage: doesn't partition data "usefully" (for queries)
- sequential scan can exploit parallelism
    - read data from multiple disks simultaneously
- index-based scan can exploit limited parallelism
    - index gives list of pages, potential parallel read
- provides no assistance for hash-based access

... Data Storage in PDBs 28/95

Hash partitioning

- hash functions: $h_N(A_i) \rightarrow NodeID$, $h_P(A_i) \rightarrow PageNum$
- well-designed hash functions *can* spread tuples uniformly
- hash-based access can work well
    - all tuples matching query hash will be on one node
- sequential scan performance depends on uniform spread
- index-on-hash works as for round-robin
- provides no assistance for range queries

... Data Storage in PDBs 29/95

Range partitioning

- uses *partitioning vector pv* to determine node for tuple
    - allocates range of partitioning attribute values to each node
- pv = [(v_0,D_i), (v_1,D_j), ... (v_h,D_m)]
    - all tuples with $A_i \le v_0$ go to D_i
    - all tuples with $v_0 < A_i \le v_1$ go to D_j, etc.
- need to choose v_i boundary points carefully
    - to ensure reasonably uniform spread of data over disks

PostgreSQL and Parallelism 30/95

PostgreSQL assumes

- shared memory space accesible to all back-ends
- files for one table are located on one disk

PostgreSQL allows

- data to be distributed across multiple disk devices

So could run on ...

- shared-memory, shared-disk architectures
- hierarchical architectures with distributed virtual memory

... PostgreSQL and Parallelism 31/95

PostgreSQL can provide

- multiple servers running on separate nodes
- application #1: high availability
  - "standby" server takes over if primary server fails
- application #2: load balancing
  - several servers can be used to provide same data
  - direct queries to least loaded server

Both need *data synchronisation* between servers

PostgreSQL uses notion of *master* and *slave* servers.

... PostgreSQL and Parallelism

32/95

High availability ...

- updates occur on master, recorded in tx log
- tx logs shipped/streamed from master to slave(s)
- slave uses tx logs to maintain current state
- configuration controls frequency of log shipping
- bringing slave up-to-date is fast (~1-2secs)

Note: small window for data loss (committed tx log records not sent)

Distributed Databases

33/95

Two kinds of distributed databases

- parallel database on a distributed architecture
  - single schema/control, data distributed over network
- independent databases on a distributed architecture
  - independent schemas/DBMSs, combined via global schema

The latter are also called *federated* databases

Distribution of data complicates tx processing ...

- potential for multiple copies of data to become inconsistent
- commit or abort must occur consistently on all nodes

... Distributed Databases

34/95

Distributed tx processing handled by *two-phase commit*

- initiating site has transaction coordinator C_i ...
  - waits for all other sites executing tx `T` to "complete"
  - sends `<prepare T>` message to all other sites
  - waits for `<ready T>` response from all other sites
  - if not received (timeout), or `<abort T>` received, flag abort
  - if all other sites respond `<ready T>`, flag commit
  - write `<commit T>` or `<abort T>` to log
  - send `<commit T>` or `<abort T>` to all other sites
- non-initiating sites write log entries before responding

... Distributed Databases

35/95

Distributed query processing

- may require query ops to be executed on different nodes
  - node provides only source of some data
  - some nodes may have limited set of operations
- needs to merge data received from different nodes
  - may require data transformation (to fit schemas together)

Query optimisation in such contexts is *difficult*.

Non-classical DBMSs

Classical DBMSs

37/95

Assumptions made in conventional DBMSs:

- data is sets of tuples; tuples are lists of atomic values
- data values can be compared precisely (via =, >, <, ...)
- filters can be described via boolean formulae
- SQL is a suitable langauage for all data management
- transaction-based consistency is critical
- data stored on disk, processed in memory
- data transferred in blocks of many tuples
- disk ↔ memory cost is most expensive in system
- disks are connected to processors via fast local bus

Modern DBMSs

38/95

Demands from modern applications

- more flexible data structuring mechanisms
- very large data objects/values (e.g. music, video)
- alternative comparisons/filters (e.g. similarity matching)
- massive amounts of data (too much to store "locally")
- massive number of clients (thousands tx's per second)
- solid-state storage (minimal data latency)
- data required globally (network latency)

Clearly, not all of these are relevant for every modern application.

... Modern DBMSs

39/95

Some conclusions:

- relational model doesn't work *for all* applications
- SQL is not appropriate *for all* applications
- hard transactions not essential *for all* applications

Some "modernists" claim that

- "for all" is really "for any"
- ⇒ relational DBMSs and SQL are dinosaurs
- ⇒ NoSQL is the new way

... Modern DBMSs

40/95

Some approaches:

- storage systems: Google FS, Hadoop DFS, Amazon S3
- data structures: BigTable, HBase, Cassandra, XML, RDF
- data structures: column-oriented DBMSs e.g. C-store
- data structures: graph databases e.g. Neo4j
- operations: multimedia similarity search e.g. Shazam
- operations: web search e.g. Google
- transactions: eventual consistency
- programming: object-relational mapping (ORM)
- programming: MapReduce
- languages: Sawzall, Pig, Hive, SPARQL
- DB systems: CouchDB, MongoDB, F1, Cstore

Scale, Distribution, Replication

41/95

Data for modern applications is very large (TB, PB, XB)

- not feasible to store on a single machine
- not feasible to store in a single location

Many systems opt for massive networks of simple nodes

- each node holds moderate amount of data
- each data item is replicated on several nodes
- nodes clustered in different geographic sites

Benefits:

- reliability, fault-tolerance, availability
- proximity ... use data closest to client
- scope for parallel execution/evaluation

Schema-free Data Models

42/95

Many new DBMSs provide *(key,value)* stores

- *key* is a unique identifier (cf. URI)
- *value* is an arbitrarily complex "object"
  - e.g. a text document (often structured, e.g. Wiki, XML)
  - e.g. a JSON object: *(property,value)* list
  - e.g. an RDF triple (e.g. `<John,worksFor,UNSW>`)
- objects may contain *key*s to link to other objects

Tables can be simulated by a collection of "similar" objects.

Eventual Consistency

43/95

RDBMSs use a strong transactional/consistency model

- if a tx commits, changes take effect "instantly"
- all tx's have a strong guarantee about data integrity

Many new DBMSs applications do not need strong consistency

- e.g. doesn't matter if catalogue shows yesterday's price

Because of distribution/replication

- update is initiated on one node
- different nodes may have different versions of data
- after some time, updates propagate to all nodes

... Eventual Consistency

44/95

If different nodes have different versions of data

- conflicts arise, and need to be resolved (when noticed)
- need to decide which node has "the right value"

Levels of consistency (from Cassandra system)

- ONE: at least one node has committed change (weakest)
- QUORUM: at least half nodes holding data have committed
- ALL: changes propagated to all copies (strongest)

MapReduce

45/95

MapReduce is a *programming model*

- suited for use on large networks of computers

- processing large amounts of data with high parallelism
- originally developed by Google; Hadoop is open-source implementation

Computation is structured in two phases:

- **Map** phase:
  - master node partitions work into sub-problems
  - distributes them to worker nodes (who may further distribute)
- **Reduce** phase:
  - master collects results of sub-problems from workers
  - combines results to produce final answer

... MapReduce

46/95

MapReduce makes use of *(key,value)* pairs

- *key* values identify parts of computation

$Map(key_1,val_1) \rightarrow list(key_2,val_2)$

- applied in parallel to all (key_1,val_1) pairs
- results with common key_2 are collected in group for "reduction"

$Reduce(key_2,list(val_2)) \rightarrow val_3$

- collects all values tagged with key_2
- combines them to produce result(s) val_3

... MapReduce

47/95

"Classic" MapReduce example (word frequency in set of docs):

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: list of aggregated partial counts
  sum = 0
  for each c in partialCounts:
    sum += c
  emit (word, sum)
```

... MapReduce

48/95

MapReduce as a "database language"

- some advocates of MapReduce have oversold it (replace SQL)
- DeWitt/Stonebraker criticised this
  - return to low-level model of data access
  - all done before in distributed DB research
  - misses efficiency opportunities affored by DBMSs
- concensus is emerging
  - SQL/MapReduce good for different kinds of task
  - MapReduce as a basis for SQL-like languages (e.g. Apache HiveQL)

Modern vs Classical

49/95

Some criticisms of the NoSQL approach:

- DeWitt/Stonebraker: *MapReduce: A major step backwards*

- [Online parody of noSQL advocates](#) (strong language warning)

  [Diagram:Pics/newdb/xtra-small.png]

Hadoop DFS

50/95

Apache Hadoop Distributed File System

- a hierarchical file system (directories & files a la Linux)
- designed to run on large number of commodity computing nodes
- supporting very large files (TB) distributed/replicated over nodes
- providing high reliability (failed nodes is the norm)

Provides support for Hadoop map/reduce implementation.

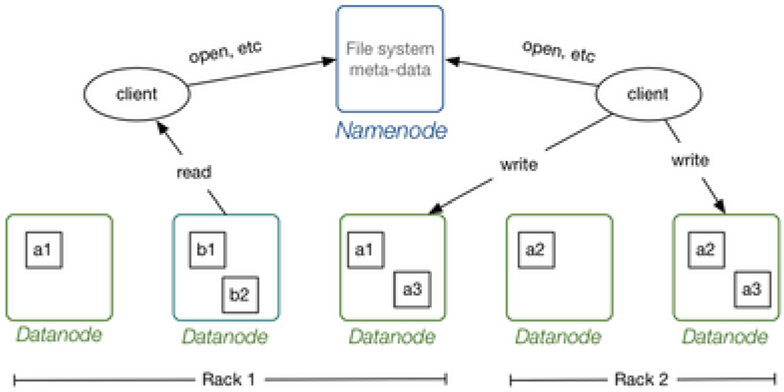
Optimised for write-once-read-many apps

- simplifies data coherence
- aim is maximum throughput rather than low latency

... Hadoop DFS

51/95

Architecture of one HDFS cluster:



... Hadoop DFS

52/95

Datanodes ...

- provide file read/write/append operations to clients
  - under instruction from Namenode
- periodically send reports to Namenode

A Hadoop *file*

- is a collection of fixed-size blocks
- blocks are distributed/replicated across nodes

Datanode → Namenode reports

- *Heartbeat* ... Datanode still functioning ok
- *Blockreport* ... list of all blocks on DataNode

... Hadoop DFS

53/95

Namenodes ...

- hold file-system meta-data (directory structure, file info)
  - e.g. file info: (filename, block#, #replicas, nodes)
  - e.g. (/data/a, 1, 2, {1,3}), (/data/a, 2, 2, {4,5}), (/data/a, 3, 2, {3,5})
- provides file open/close/rename operations to clients
- determine replication and mapping of data blocks to DataNodes
- select Datanodes to serve client requests for efficient access

- e.g. node in local rack > node in other rack > remote node

Namenode knows file ok if all relevant Datanodes sent Bockreport

- if not ok, replicate blocks on other Datanodes & update meta-data

---

Two Case Studies

Consider two variations on the DBMS theme ...

Column Stores

- still based on the relational model
- but with a variation in how data is stored
- to address a range of modern query types

Graph Databases

- based on a graph model of data
- emphasising explicit representation of relationships
- relevant to a wide range of application domains

Column Stores

(Based on material by Daniel Abadi et al.)

Column Stores

Column-oriented Databases (CoDbs):

- are based on the relational model
- store data column-by-column rather that row-by-row
- leading to performance gains for analytical applications

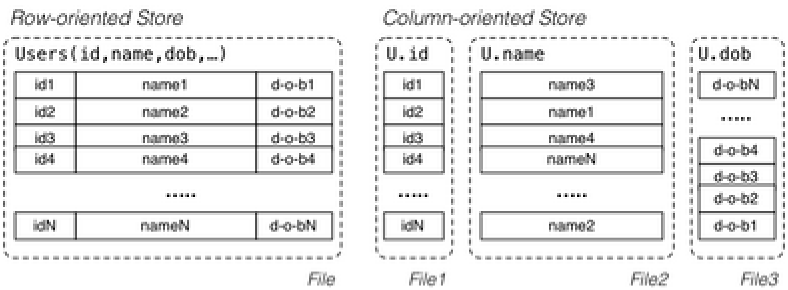
Ideas for CoDbs have been around since the 1970's

Rose to prominence via Daniel Abadi's PhD thesis (MIT, 2008)

Commercial systems have now been developed (e.g. Vertica)

... Column Stores

File structures for row-store vs column-store:



Values in individual columns are related by extra tuple id (cf. oid)

... Column Stores

Stored representation of logical (relational) tables

- each table is stored as a set of projections (slices)

- each projection consists of a different set of columns
- each column appears in at least one projection
- "rows" can be ordered differently in each projection

Example: Enrolment(course,student,term,mark,grade)

- *projection*$_1$: (course,student,grade) ordered by course
- *projection*$_2$: (term,student,mark) ordered by student
- *projection*$_3$: (course,student) ordered by course

Rows vs Columns

59/95

Workload for different operations

- *insert* requires more work in CoDbs
  - row: update one page; column: update multiple pages
- *project* comes "for free" in CoDbs
  - row: extract fields from each tuple; column: merge columns
- *select* may require less work in CoDbs
  - row: read whole tuples; column: read just needed columns
- *join* may require less work in CoDbs
  - row: hash join; column: scan columns for join attributes

... Rows vs Columns

60/95

Which is more efficient depends on mix of queries/updates

- RDBMSs are, to some extent, write-optimized
  - effective for OLTP applications (e.g. ATM, POS, ...)
- when RDBMSs might be better ...
  - when query requires all attributes
  - might read more data, but less seek-time (multiple files)
- when CoDbs might be better ...
  - smaller intermediate "tuples"
  - less competition for access to pages (locking)

... Rows vs Columns

61/95

Storing sorted columns leads to

- potential for effective compression
  - compression ⇒ more projections in same space
  - no need to compress all columns (if some aren't "compressible")
- sorted data is useful in some query evaluation contexts
  - e.g. terminating scan once unique match found
  - e.g. sort-merge join

Only one column in each projection will be sorted

- but if even one projection has a column sorted how you need ...

Query Evaluation in CoDbs

62/95

Projection is easy if one slice contains all required attributes.

If not ...

- sequential scan of relevant slices in parallel
- combine values at each iteration to form a tuple

Example: `select a,b,c from R(a,b,c,d,e)`

```
Assume: each column contains N values
for i in 0 .. N-1 {
   x = a[i]   // i'th value in slice containing a
```

```
   y = b[i]   // i'th value in slice containing b
   z = c[i]   // i'th value in slice containing c
   add (x,y,z) to Results
}
```

... Query Evaluation in CoDbs

63/95

If slices are sorted differently, more complicated

- scan based on tid values
- at each step, look up relevant entry in slice

Example: `select a,b,c from R(a,b,c,d,e)`

```
Assume: each column contains N values
for tid in 0 .. N-1 {
   x = fetch(a,tid)  // entry with tid in slice containing a
   y = fetch(b,tid)  // entry with tid in slice containing b
   z = fetch(c,tid)  // entry with tid in slice containing c
   add (x,y,z) to Results
}
```

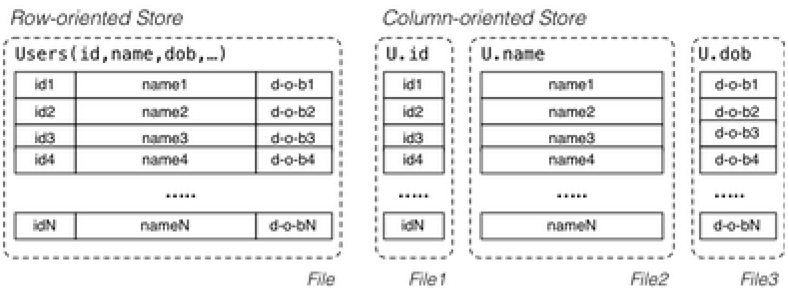
Potentially slow, depending on how `fetch()` works.

... Query Evaluation in CoDbs

64/95

For remaining discussion, assume

- each slice has 1 attribute, and a[i].tid = b[i].tid = c[i].tid



... Query Evaluation in CoDbs

65/95

Consider typical multi-attribute SQL query

`select a,b,c from R where b > 10 and d < 7`

Query operation on individual column is done in one slice

Mark index of each matching entry in a bit-vector

Combine (AND) bit-vectors to get indexes for result entries

For each index, merge result entry columns into result tuple

Known as *late materialization*.

... Query Evaluation in CoDbs

66/95

Example: `select a,b,c from R where b = 5`

```
// Assume: each column contains N values
matches = all-zero bit-string of length N
for i in 0 .. N-1 {
   x = b[i]  // i'th value in b column
```

```
    if (x == 5)
        matches[i] = 1  // set bit i in matches
}
for i in 0 .. N-1 {
    if (matches[i] == 0) continue
    add (a[i], b[i], c[i]) to Results
}
```

Fast sequential scanning of small (compressed?) data

... Query Evaluation in CoDbs

67/95

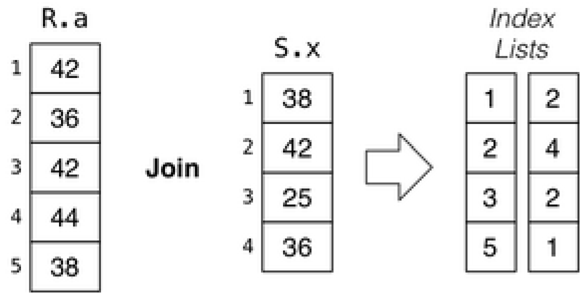
Example: `select a,b,c from R where b>10 and d<7`

```
// Assume: each column contains N values
matches1 = all-zero bit-string of length N
matches2 = all-zero bit-string of length N
for i in 0 .. N-1 {
   if (b[i] > 10) matches1[i] = 1
   if (d[i] < 7) matches2[i] = 1
}
matches = matches1 AND matches2
for i in 0 .. N-1 {
   if (matches[i] == 0) continue
   add (a[i], b[i], c[i]) to Results
}
```

... Query Evaluation in CoDbs

68/95

Join on columns, set up for late materialization



Note: the left result column is always sorted

... Query Evaluation in CoDbs

69/95

Example:
```
select R.a, S.b
from   R join S on R.a = S.x

// Assume: N tuples in R, M tuples in S
for i in 0 .. N-1 {
   for j in 0 .. M-1 {
      if (a[i] == x[j])
         append (i,j) to IndexList
   }
}
for each (i,j) in IndexList {
   add (a_R[i], b_S[j]) to Results
}
```

... Query Evaluation in CoDbs

70/95

Aggregation generally involves a single column

- multiple aggregations could be carried out in parallel

E.g.

```
select avg(mark), count(student) from Enrolments
```

Operations involving groups of columns

- may require early materialization ⇒ slower

Graph Databases

(Based on material by Markus Krotzsch, Renzo Angles, Claudio Gutierrez)

Graph Databases

72/95

Graph Databases (GDbs):

- DBMSs that use *graphs* as the data model

But what kind of "graphs"?

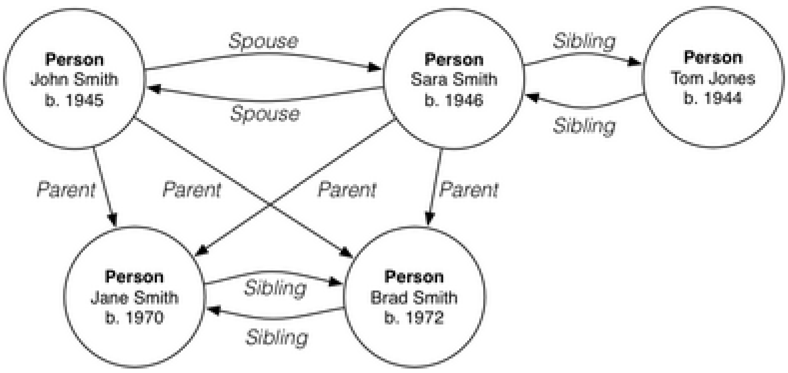
- all graphs have nodes and edges, but are they ...
- directed or undirected, labelled or unlabelled?
- what kinds of labels? what datatypes?
- one graph or multiple graphs in each database?

Two major GDb data models: RDF, Property Graph

... Graph Databases

73/95

Typical graph modelled by a GDb



Graph Data Models

74/95

RDF = Resource Description Framework

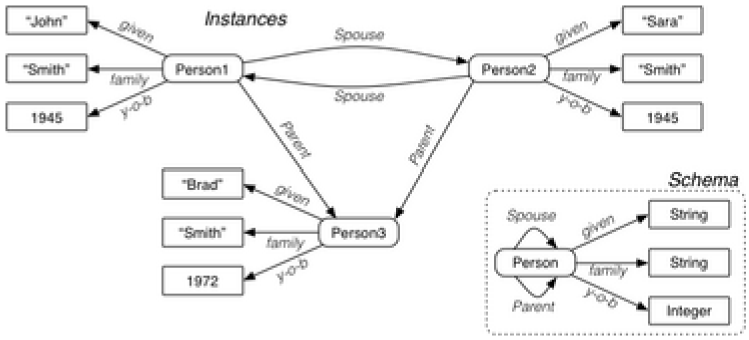
- directed, labelled graphs
- nodes have identifiers (constant values, incl. URIs)
- edges are labelled with the relationship
- can have multiple edges between nodes (diff. labels)
- can store multiple graphs in one database
- datatypes based on W3C XML Schema datatypes

Data as triples, e.g. <Person1,given,"John">, <Person1,parent,Person3>

RDF is a W3C standard; supported in many prog. languages

... Graph Data Models
75/95

RDF model of part of earlier graph:



... Graph Data Models
76/95

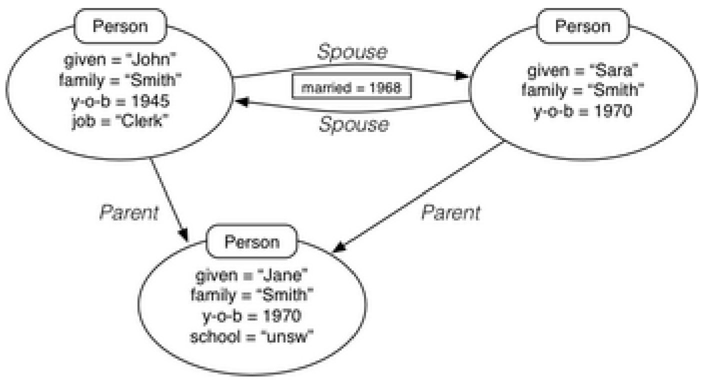
Property Graph

- directed, labelled graphs
- properties are (key/label, value) pairs
- nodes and edges are associated with a list of properties
- can have multiple edges between nodes (incl same labels)

Not a standard like RDF, so variations exist

... Graph Data Models
77/95

Property Graph model of part of earlier graph:



GDb Queries
78/95

Graph data models require a graph-oriented query framework

Types of queries in GDbs

- node properties (like SQL `where` clauses)
  - e.g. is there a Person called John? how old is John?
- adjacency queries
  - e.g. is John the parent of Jane?
- reachability queries
  - e.g. is William one of John's ancestors?
- summarization queries (like SQL aggregates)
  - e.g. how many generations between William and John?

... GDb Queries
79/95

Graphs contain arbitrary-length paths

Need an expression mechanism for describing such paths

- path expressions are regular expressions involving edge labels
- e.g. `L*` is a sequence of one or more connected `L` edges

GDb query languages:

- SPARQL = based on the RDF model (widely available via RDF)
- Cypher = based on the Property Graph model (used in Neo4j)

Example Graph Queries

80/95

Example: Persons whose first name is James

SPARQL:

```
PREFIX p: <http://www.people.org>
SELECT ?X
WHERE { ?X p:given "James" }
```

Cypher:

```
MATCH (person:Person)
WHERE person.given="James"
RETURN person
```

... Example Graph Queries

81/95

Example: Persons born between 1965 and 1975

SPARQL:

```
PREFIX p: <http://www.people.org/>
SELECT ?X
WHERE {
 ?X p:type p:Person . ?X p:y-o-b ?A .
 FILTER (?A ≥ 1965 && ?A ≤ 1975)
 }
```

Cypher:

```
MATCH (person:Person)
WHERE person.y-o-b ≥ 1965 and person.y-o-b ≤ 1975
RETURN person
```

... Example Graph Queries

82/95

Example: pairs of Persons related by the "parent" relationship

SPARQL:

```
PREFIX p: <http://www.people.org/>
SELECT ?X ?Y
WHERE { ?X p:parent ?Y }
```

Cypher:

```
MATCH (person1:Person)-[:parent]->(person2:Person)
RETURN person1, person2
```

... Example Graph Queries

83/95

Example: Given names of people with a sibling called "Tom"

SPARQL:

```
PREFIX p: <http://www.people.org/>
SELECT ?N
```

```
WHERE { ?X p:type p:Person . ?X p:given ?N .
        ?X p:sibling ?Y . ?Y p:given "Tom"  }
```

Cypher:

```
MATCH (person:Person)-[:sibling]-(tom:Person)
WHERE tom.given="Tom"
RETURN person.given
```

... Example Graph Queries

84/95

Example: All of James' ancestors

SPARQL:

```
PREFIX p: <http://www.socialnetwork.org/>
SELECT ?Y
WHERE { ?X p:type p:Person . ?X p:given "James" .
        ?Y p:parent* ?X }
```

Cypher:

```
MATCH (ancestor:Person)-[:parent*]->(james:Person)
WHERE james.given="James"
RETURN DISTINCT ancestor
```

Course Review + Exam

Syllabus

86/95

View of DBMS internals from the bottom-up:

- storage subsystem (disks,pages)
- buffer manager, representation of data
- processing RA operations (sel,proj,join,...)
- combining RA operations (iterators/execution)
- query translation, optimization, execution
- transactions, concurrency, durability
- non-classical DBMSs

Exam

87/95

Tuesday 27 August, 1.45pm - 5pm, (1.45 = reading time)

Held in CSE Labs (allocations posted in Week 11).

All answers are typed and submitted on-line.

Environment is similar to Vlab.

Learn to use the shell, a text editor and on-screen calculator.

... Exam

88/95

Resources available during exam:

- exam questions (collection of web pages)
- PostgreSQL manual (collection of web pages)
- C programming reference (collection of web pages)
- course notes (HTML version from Course Notes)

No access to any other material is allowed.

No network access is available (e.g. no web, no email, ...)

... Exam

89/95

Tools available during the exam

- C compiler (`gcc`, `make`)
- text editors (`vim`, `emacs`, `gedit`, `nedit`, `nano`, ...)
- on-screen calculators (`bc`, `gcalctool`, `xcalc`)
- all your favourite Linux tools (e.g. `ls`, `grep`, ...)
- Linux manual (`man`)

What's on the Exam?

90/95

Potential topics to be examined ...

- ~~A - Course Introduction, DBMS Revision,~~ PostgreSQL
- B - Storage: Devices, Files, Pages, Tuples, Buffers, Catalogs
- C - Cost Models, Implementing Scan, Sort, Projection
- D - Implementing Selection on One Attribute
- E - Implementing Selection on Multiple Attributes
- F - Similarity-based Selection (only first 15 slides)
- G - Implementing Join
- H - Query Translation, Optimisation, Execution
- I - Transactions, Concurrency, Recovery
- J - Non-classical DBMSs

... What's on the Exam?

91/95

Questions will have the following "flavours" ...

- write a small C program to do V
- describe what happens when we execute method W
- how many page accesses occur if we do X on Y
- explain the numbers in the following output
- describe the characteristics of Z

There will be **no** SQL/PLpgSQL code writing.

You will **not** have to modify PostgreSQL during the exam.

Exam Structure

92/95

There will be 8 questions

- 2 x C programming questions (40%)
- 6 x written answer questions (60%)

Reminder:

- exam contributes 60% of final mark
- hurdle requirement: must score > 24/60 on exam

Special Consideration

93/95

Reminder: this is a one-chance exam.

- attendance at the Exam is treated as "I am fit and well"
- subsequent claims of "I failed because I felt sick" are ignored

If you're sick, get documentation and do not attend the exam.

Special consideration requests must clearly show

- how *you* were personally affected
- that your ability to study/take-exam was impacted

Other factors are not relevant (e.g. "I can't afford to repeat")

Revision

94/95

Things you can use for revision:

- past exams
- theory exercises
- prac exercises
- course notes
- textbooks

Pre-exam consultations leading up to exam (see course web site)

Note: I'm away on August 18-20 inclusive (no email)

And that's all folks ...

95/95

End of COMP9315 19T2 Lectures.

Good luck with the exam ...

And keep on using PostgreSQL ...

Produced: 8 Aug 2019