Database Systems
CSE 414

Lecture 29: Final Review

Announcements
• HW8 due tonight 11pm

Final Exam
• Next Thursday, Dec. 14th, 2:30-4:20
• This room
• Closed books, no phones, no computers
• Allow 2 pages of notes (both sides, 8+pt font)
  – but focus of the test will not be memorization

Course Topics
1. Relational Data
2. DB Applications: Design & Implementation
3. Semi-structured Data
4. DBMS Implementation
5. Big Data Systems

Relational Data

1a. Relational Data Model
• tables with schemas
  – types for attributes
  – primary, secondary, and foreign keys
  – other constraints
• set semantics
  – each tuple is either in the table or not
1b. Relational Queries

- Relational query = expressible in standard RA
  - RA = Datalog + neg, also expressible with SQL
- Simple SELECT-FROM-WHERE is a subset
  - Includes joins, but not subqueries
  - Always monotone, while RA isn’t (e.g. set difference)
- Extended RA adds grouping & aggregation
  - (Also uses bag semantics)
- Datalog adds recursion

1c. Datalog

- Data comes from facts and rules
  - \( P(a_1, \ldots, a_n) \).
  - \( Q(a_1, \ldots, a_n) :- R_1(a_i, b_k, \ldots), R_2(a_j, b_l, \ldots), \ldots \).
- Head is a fact iff there is some way to set \( b_k \)'s so that all terms in the body are facts
  - Variables only appearing in body \( (b_k)'s \) are existential
- Can be translated to SQL
  - Must be possible, as Datalog is equivalent to RA
  - But we didn’t discuss the details...

2a. DB Design Process

Conceotual Model:

<table>
<thead>
<tr>
<th>Relational Model:</th>
<th>Conceptual Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables + constraints</td>
<td>And also functional dep.</td>
</tr>
<tr>
<td>Also functional dep.</td>
<td></td>
</tr>
</tbody>
</table>

Normalization:

<table>
<thead>
<tr>
<th>Conceptual Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliminates anomalies</td>
</tr>
<tr>
<td>Conceptual Schema</td>
</tr>
</tbody>
</table>

Physical Storage Details:

<table>
<thead>
<tr>
<th>Physical Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Schema</td>
</tr>
</tbody>
</table>

2a. DB Design Process

- E/R Diagrams
  - (Weak) entity sets, relations, & subclasses
  - Map each to relations
    - Multiple ways to do this...
      - Only need to know the approach from class
  - Design principles:
    - Model accurately
    - Neither too few nor too many entities
2a. DB Design Process

- **Constraints**
  - key, single-value, referential & other constraints
  - other includes, e.g., positivity and non-null constraints
- **Normalization**
  - eliminates anomalies
  - redundancy, update, and deletion anomalies
  - are indicated by "bad" functional dependencies
  - apply BCNF decomposition to remove them
  - these decompositions are never lossy (others can be)

2b. DB Application Implementation

- **JDBC**
  - connect to DB from Java
  - send SQL statements
  - use transactions
- **3-tiered architecture for web applications**
  - usually JSON data between web server & browser/phone
  - why not use JSON to the DB too?
    - otherwise, we need to translate JSON to relational

3-Tiered Architecture

3a. Semi-structured Data Model

- tree structured data: JSON, XML, etc.
- data is self-describing
  - so schema is not necessary
- can choose amount of structure (in AsterixDB)
  - partial constraints on shape of data
  - open vs. closed types
- NFNF data
  - could put entire data in one row (mondial)
- easy to map relation to JSON, but not opposite
3b. Semi-structured Queries

- new concepts
  - **unnesting**: join with contents of list-valued column
  - **nesting**: make list from results of subquery
  - each is a new operator for logical query plans

- dealing with heterogeneous data needs work
  - often CASE WHEN … for different types
  - requiring more structure makes queries easier, but adding data becomes harder
    - (this work has to be done somewhere)

4a. Storage & Indexing

- B+ tree & hash indexes
  - B+ tree index allows searching by key prefixes also
- understand when an index can be used
  - (separate question from whether it improves performance)
- clustered vs. unclustered
  - clustered always speeds up query, but only one index per table can be clustered
  - unclustered only speeds up if <1% tuples match

4b. Query Optimization

- main cost is disk access
- many logical plans, many physical plans
  - logical plans are RA expressions with desired result
  - physical plans include, e.g., choice of join algorithm
    - hash, sorted merge, and (block refined) nested loop joins
- cost of many operations depends on selectivity
- optimization problem is hard
  - saw SQL Server does poorly in homework problems
- realistic goal is to avoid really bad plans

4c. Transactions

- Goal: to allow many clients to run simultaneously
  - OLTP workload: lots of clients with small read/writes
- need to provide ACID properties
  - atomic: execute all SQL statements or none
  - consistent: finish with all constraints satisfied
  - isolation: behavior same as if one-at-a-time use
  - durable: committed result are permanent (’til changed)
- consistency maintained by checking constraints
- durability maintained by writing to disk(s)
4c. Transactions II

- isolation achieved through serializable schedules
  - serializable means same behavior as a serial schedule
  - conflict serializable means non-conflicting read/writes
    can be swapped to make schedule serial
    - stronger than (so implies) serializable
- locks ensure conflict serializability if 2PL used
  - multiple read locks, only one write lock
    - becomes 4 types in SQLite (a good design)
  - lock granularity from (parts of) rows to tables to DB
  - ...

4c. Transactions III

- strict 2PL: no unlocks before commit/rollback
  - needed for isolation if txns can roll back
  - can produce deadlocks (as seen in homework)
  - need more to prevent phantom rows
    - phantom is a new row that shows up in a table
    - predicate locks are one solution (but expensive)
- multi-version concurrency control is alternative
- default isolation level is usually not serializable
  - faster perf but harder to write app (i.e., bugs likely)

5a. NoSQL Systems

- goal to support heavy OLTP workloads
- provides simplified data model
  - key-value pairs, documents, or extensible records
- limited support for transactions
  - usually pair/document/record level
  - (some support for record groups… all on one node)
- partition data across nodes for scale
- replicate data to survive node failures

5b. Parallel Processing Systems

- for OLAP workloads (big reads, no txns)
- MapReduce
  - programming model is one-to-many map function, shuffle sort (grouping), one-to-many reduce function
  - no built-in RA operators
    - but easy to implement, as shuffle sort is provided
  - stores intermediate data on disk
    - reasonable if input/output is also to disk (otherwise too slow)
  - deals with stragglers by running backup map tasks

5b. Parallel Processing Systems II

- Spark/Scala
  - executes a dataflow pipeline using many nodes
  - Google Dataflow & Hyracks (AsterixDB) do same
    - each provides extended RA operators
    - Spark handles failure by re-computing, not replicating
- Spark SQL
  - map SQL -> extended RA -> dataflow pipeline
  - same approach can be used on any dataflow engine
5b. Parallel Processing Systems III

- Existing systems do not optimize well
  - none does real cost-based optimization
  - Spark only performs small, syntactic optimizations
    - one exception: choice of parallel vs. broadcast join
  - Spark has no indexes
  - AsterixDB has indexes, but no statistics
  - all require manual tuning
    - saw this with AsterixDB on homework
- PageRank

---

5c. Parallel Databases

- support both OLTP and OLAP
- goal: more nodes => faster or allow more data
  - speed up or scale up
- different architectures
  - shared memory (SQL Server etc.): limited scale
  - shared disk (mostly Oracle): limited scale
  - shared nothing: really scales (so our focus)
    - won out in academic research (started in 1980s)
    - basis for parallel processing systems (see previous slides)

---

5c. Parallel Databases II

- Partition data across nodes (hash, range, etc.)
- Query evaluation
  - only one new element: reshuffle
    - move tuples to nodes based on values in certain columns
    - basically same as shuffle sort of MapReduce
    - use to implement all extended RA operations
  - linear speed up or scale up in principle
  - in practice, stragglers are a problem (MapReduce tries to discover and redo the tasks the stragglers are working on)
  - new problem: skewed data
    - may not all fit in memory of one node

---

5c. Parallel Databases III

- AsterixDB is the closest we have seen to this
  - came out of parallel DB community
  - executes OLAP queries as in parallel processing
  - but only has record-level transactions as in NoSQL
    - (more OLTP than parallel processing systems though)
- More complete systems in the near future
  - see also Google Spanner, Microsoft Cloud DB

---

5. SQL

- CREATE TABLE ...
  - PRIMARY KEY, UNIQUE, FOREIGN KEY
  - CHECK (constraints) on columns or tuples
- CREATE [CLUSTERED] INDEX ... ON ...
- INSERT INTO ...
- UPDATE ... SET ... WHERE ...
- DELETE FROM ... WHERE ...
5. SQL (cont.)

- SELECT …
  - JOINs: inner vs. outer, natural
  - GROUP BY, sum, count, avg, etc.
  - ORDER BY
- SET ISOLATION LEVEL …
- BEGIN TRANSACTION
- COMMIT / ROLLBACK