Final Exam Review

Last Lecture
R&G - All Chapters Covered

The end crowns all,
And that old common arbitrator, Time,
Will one day end it.

William Shakespeare.
Troilus and Cressida.

Topics Covered

- **Relational Model**
- **Disks, Files, Buffers, Indexes**
- **Languages: Algebra, Calculus, SQL**
- **Query Processing: Itertaors, Sorting, Hashing, Join Algs, Parallelization**
- **Query Optimization**
- **Schema Refinement and Normalization**
- **Search Engines: Boolean search, Ranking (TFxIDF and PageRank)**
- **Concurrency Control**
- **Crash Recovery**

NOT on the final

- **Hash indexes**
- **Ruby on Rails**
- **MapReduce**
- **"Division" operator in Relational Algebra**
- **Normalization and Functional Dependencies**
- **Optimistic/Timestamp/Multiversion concurrency control**
- **Fixing the Phantom problem**
  - Though you should know what it is
- **Web crawler design**

Overview

- **Purpose of this course: give students both**
  - An understanding of what databases and data-centric systems do, why we use them, and how to use common databases efficiently, and
  - An understanding of how databases and search engines work internally.

Introduction

- **What are databases?**
- **Data models**
- **What does a DBMS provide that the OS does not?**
  - Levels of Abstraction, Data Independence, Concurrency Control, Crash Recovery, etc.

Introduction (cont)

- **schemas & data independence**
  - conceptual schema
  - physical schema
  - external schema (view)
  - logical & physical data independence

The Relational Data Model Basics

- **Components of the model:**
  - Relations, Attributes, Tuples
- **SQL Data Definition Language**
- **Integrity Constraints**
  - how do they come into being?
  - understand what you can learn from schema vs. instance!
- ***Referential Integrity***
  - a state which holds when all foreign key constraints are enforced

Relational Model (cont)

- **Keys, Primary Keys, Foreign Keys, Candidate Keys**
- ***Foreign key* : Set of fields in one relation that is used to `refer' to a tuple in another relation.**
  - Must correspond to primary key of the second relation.
  - Like a `logical pointer'.
- **A set of fields is a *key* for a relation if :**
  1. No two tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
  - Part 2 false? A *superkey*.
  - If there's >1 key for a relation, all are *candidate keys.* One of the candidate keys is chosen to be *primary key*.
- **E.g., *sid* is a key for Students. (What about *name*?) The set {*sid, gpa*} is a superkey.**

The Entity-Relationship Data Model

- **Typically also tool for conceptual design**
- **Components of the model:**
  - Entities, Attributes, Relationships
  - Cardinality of relationships, key constraints
  - Participation Constraints
  - Weak entities
- **Be able understand an E-R diagram**
- **Be able to translate an E-R Diagram into Relational schema**
- **Be able to modify an E-R diagram**

Memory Management

- **Hierarchy of storage: RAM, Disk, Tape**
- **Advantages/disadvantages of different types of storage**
- **Buffer management**
  - You should know basics from project!
    - Understand data structures required
    - notions of *replacement*, *dirty* pages, *pinning* pages
  - Understand different replacement policies
    - LRU, MRU, CLOCK
    - be able to simulate each, understand pros, cons!

Memory Management (cont)

- **Organizing records in pages**
  - Fixed & variable-length fields in tuples
    - 2 alternatives for variable-length fields
  - Fixed & variable-length tuples on pages
    - know what a RID is, how it interacts with page layout
    - know details of "slotted page" with slot directory
- **Organizing pages in files**

File Organization

- **Different file organizations: heap files, sorted files, hashed files**
  - be able to compute costs of ops over each!
- **Access costs for different organizations**

File Organization: Indexes

- **Understand search keys (vs. key constraints!)**
- **3 alternatives for data entries**
  - ① Data record with key value **k**
  - ② <**k**, rid of data record with search key value **k**>
  - ③ <**k**, list of rids of data records with search key **k**>
  - – Choice of alternative orthogonal to indexing technique!

Tree Structured Indexes

- **Trees do *range* and *equality* search**
- **ISAM, rules for adding and removing entries**
- **B-Trees, rules for adding entries**
  - – You do not have to do "coalescing" on B+-tree deletion

External Sorting, Hashing

- **How to sort any file using 3 memory Pages**
- **How to sort/hash in as few passes given some amount of memory**
- **Relationship between buffers of memory, size of file, and number of passes to merge**
- **Duality of sort and hash**
- **Application to duplicate elimination, group by**

Relational Algebra

- **Query language operating on relations**
- **Know the operators!**
  - – σ, π, $\times$, $\cup$, $\cap$, -, ρ,
  - – know schemas of output relations
  - – know varieties of joins
    - (conditional vs. equi vs. natural), division
  - – be able to express complex ops in terms of simple ones
- **Use relational algebra to express queries written in English, and vice versa**

Relational Calculus

- **Declarative query language for tuples**
- **formulas, operators, in, not, there-exists, for-all**
- **Use relational calculus to express simple queries written in English, and vice versa**

SQL

- **DDL: "Create Table"**
- **DML: Delete From, Insert Into, Update**
- **Basic Query:**

  select <targets>
  from <relations>
  where <qualification>
- **Use of Distinct clause**
- **Set operations: Union, Except, Intersect**

SQL Query Language

- **basic queries**
- **conceptual evaluation**
- **range variables**
- **expressions**
- **strings**
- **Union, Intersect, In, Except, Exists**
- **nested queries, correlated and not**
- **set comparison**
- **Aggregation**
  - operators: Count, Avg, Any
  - Group By, Having

SQL Continued

- **Aggregation**
  - Count, Sum, Avg, Max, Min
  - Group By, Having clauses
- **Nested Queries**
  - in Where or From Clause
  - set comparison:
    - In, Exists, Unique
    - *op* Any, *op* All
  - correlated vs. uncorrelated

Implementation of Relational Operators

- **Important Operators:**
  - Selection
  - Projection
  - Set operations
    - Set-Difference, Union
  - Aggregation
  - Join
- **Understand cost estimation, selectivity**

Selection

- **with no index, scan entire relation**
- **with clustered index, use index**
- **with unclustered index, sort rids**
- **hash index only good for equality selection**
- **with multiple selection conditions, either**
  - scan entire table testing all conditions
  - use index on most restrictive conditon first, scan result for other conditions
  - use index for each condition, do set intersection on RIDs

Projection

- **Hard part: removing duplicates (if necessary)**
- **Can remove duplicates by sorting or hashing**
- **If index contains projected attr, can do index-only scan**

Set Operations & Aggregation

- **Set operations:**
  - Intersection, cross product treated like joins
  - Union (Distinct) and Set Difference both involve finding duplicates between two sets, treat similar to Project
- **Aggregation**
  - without Group By, must scan entire relation
  - with Group By, must sort, then scan

Joins

- **Two relations: inner N and outer M**
- **Simple Nested Loops:**
  - for each outer tuple, scan inner for matches
  - cost:M + #tuples in M * N
- **Paged Nested Loops**
  - for each page in M, scan inner for matches to any tuple in that M page
  - cost: M + M*N
- **Blocked Nested Loops**
  - like paged, except put as much of M in memory as possible, leaving 1 page for N and 1 page for output
  - cost: M + (M/(block size)) * N

Joins (cont)

- **Indexed Nested Loops**
  - for each tuple in M, use index to find matches in N
  - cost: M + #tuples in M * cost to use index to get tuples
- **Sort-Merge Join**
  - Sort each table, merge finding like values,
  - can be bad if many duplicates
  - cost: M log M + N log N + M + N
- **Hash Join**
  - partition both relations into buckets, read in one bucket from M at a time, match with elements from same bucket in N
  - cost: partioning 2*(M + N) plus matching (M + N)

Query Optimization

- **Some ops are commutative, associative**
- **May change the order of many ops in query**
- **Dramatic changes in cost depending on op order**
- **"Query Plan" - a tree of ops indicating ops, order**
- **Ideally find optimal plan**
- **In reality avoid terrible plans**

Optimizer Implementation

- **need iterator interface so each op passes tuples on to the next**
- **need cost estimator to determine costs of different plans**
  - Reduction Factor of ops
- **need statistics and catalogs to estimate costs**

System R Optimizer in action

- **convert SQL to relational algebra**
- **find alternate plans**
  - for each relation, consider all access paths
  - for multiple relations, consider different join algorits, access paths
  - for join orders, only consider left-deep trees

Concurrency Control

- **Transaction: basic unit of operation**
  - made up of reads and writes
- **Goal: ACID Transactions**
- **A & D are provided by Crash Recovery**
- **C & I are provided by Concurrency Control**
- **Bottom line: reads and writes for various transactions MUST be ordered such that the final state of the database is the same as *some serial ordering of the transactions***

Approaches to Concurrency Control

- **2PL - all objects have Shared and eXclusive locks**
  - once one lock is released, no more locks may be acquired
  - Strict 2PL: don't release locks until commit time
  - Conservative 2PL: acquire all locks at start, release all at end
- **Locking issues**
  - must either prevent or detect deadlock
  - may want multiple granularity locks (table, page, record) using IS, IX, SIX, S, X locks *(check compatibility matrix!)*
  - locking in B-trees usually not 2PL
  - phantom problem: locking all records of a given criteria (e.g., age > 20)

Crash Recovery

- **ACID - need way to ensure A & D**
- **We studied approach of Aries system**
- **Buffer management Steal, no Force**
- **Every Write to a page is first logged in WAS**
  - log record is in stable storage before data page on disk
  - log record has Xact#, before value, after value
- **Checkpoints record which pages dirty, which XActs running**

Transaction Commit

- **write Commit record to log**
- **flush log tail to stable storage**
- **remove Xact from Xact table**
- **write End record to log**

Transaction Abort

- **write Abort record to log**
- **go back through log, undoing each write (and add CLR to log)**
- **when done, write End record to log**

Crash Recovery - 3 phases

- **Analysis: starting from checkpoint, go forward in the log to see:**
  - what pages were dirty
  - what transactions were active at time of crash
- **Redo: start from oldest transaction that wrote to a dirty page, and redo all writes to dirty pages.**
- **Undo: start at the end of the log (time of crash), work backward undoing all writes made by transactions that were active at time of crash**

And finally…