15-721 Database Management Systems

System R and the Relational Model

Instructor: Anastassia Ailamaki
http://www.cs.cmu.edu/~natassa

Detailed Roadmap

- Intro
  - Codd’s paper
  - System R - design
  - System R - evaluation

The Roots

- Codd (CACM’70): Relational Model
- Bachman (Turing Award, 1973): DBTG
  - (network model based on COBOL)
- SIGMOD 1975: The Great Debate
  - pros and cons??
The Roots

CODASYL:  Relational:
  - RL too much math  - DBTG complicated
  - Implementation  - No easy set queries
  - OLTP <-> operators  - No semantics

Late 70's: Relational Model wins

Relational Prototypes

- SQL, Quel (user-friendlier than Rel. algebra)
- Performance issue addressed

Relational Prototypes

<table>
<thead>
<tr>
<th>System R</th>
<th>INGRES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM SJ, 1974-78</td>
<td>UCB 1973-77</td>
</tr>
<tr>
<td>compiler</td>
<td>Interpreter</td>
</tr>
<tr>
<td>RDS/RSS links</td>
<td>Unix FS (no recovery!)</td>
</tr>
<tr>
<td>Recovery scheme</td>
<td>16-bit PDP-11</td>
</tr>
<tr>
<td>No hashing</td>
<td></td>
</tr>
</tbody>
</table>
Impact

System R
- ESVAL / HP Allbase, IDMS/SQL,
- Oracle, DB2, SQL/DS
- Query optimization
- Compilation

INGRES
- INGRES Corp., Britton-Lee IDM, Sybase
- Clean QUEL
- Queries for views
- Protection, integrity

But: both systems unfaithful to Rel. Model:
- allow duplicate records
- No notion of domain or primary key

Detailed Roadmap

- Intro
  - Codd’s paper
    - System R - design
    - System R - evaluation

Codd, CACM’70

Goals:
- (logical + physical) Data independence
  - Ordering (sorted vs. raw)
  - Indexing (existence or not)
  - Access path dependency
  - Avoid inconsistencies
(putting things in context: DBTG)

DBTG = CODASYL = Network model:
- repeating groups
- records (eg., ‘employee’, ‘department’)
- sets (eg., ‘employee works in a department’)
  ['marketing’, {John, Mary, Mike}]
  ['sales’, {Peter, Tom}]
...

(putting things in context: DBTG)

QL: ‘fetch’, ‘fetch next’, ‘fetch within parent’
- Fast, for suitable queries;
- bad for rest
- even worse, apps break if schema changes

Salvation:
- Everything is a table - no ‘DBTG sets’, no repeating groups
- In detail:
The Relational Model
- Relation (dom, … dom)
- \( R (s_1, \ldots, s_n) \) \( \subseteq S_1 \times \ldots \times S_n \)
- Rows
  - Distinct
  - Ordering doesn’t matter
- Columns
  - Order matters
  - Order + labels = unique identification
- Primary key, foreign key

Codd, CACM’70 (cont.)
- First Normal Form (1NF)
  - Simple domains only --> attributes
  - No repeating groups
  - Advantages/disadvantages?
- Language
  - Declaration of relations (today: DDL)
  - Queries (today: DML)
  - Insertion/deletion/update

Operations and Rules
- Set operations on relations
- Projection \( \pi_{12} (R(s_1, s_2, s_3)) = R'(s_1, s_2) \)
- Join \( R \bowtie S \)
- Composition \( \pi_{13} (R \bowtie S) \)
- Restriction (selection with AND, OR)
('Restriction')

\[ R' = R_{(2,3)} \mathbin{\mid}_{(1,2)} S \]
i.e., give the \((2,3)\) tuples of ‘\(R\)’ that match a
tuple from ‘\(S\)’

Formally: \(R'\) is the maximal subset of \(R\) s.t.
projection\((2,3)\)(\(R'\)) = projection\((1,2)\)(\(S\))

[hence CODASYL’s complaints!]

Operations and Rules - cont’d

- Redundancy (no derivable relations)
  - ‘strong’ (an existing table is a projection of
    some other)
  - ‘weak’ (……… of some join)
- [either way, the yet-to-be-invented Functional
  Dependencies would capture them]

- Consistency
  - [the penalty for redundancy: need to check]

Reminders

Goals:

- (logical + physical) Data
  independence
- Avoid inconsistencies
Today:

Five fundamental operators, for rel. algebra
- union
- difference
- selection
- projection
- cartesian product

Today:

For Inconsistencies:
- Functional Dependencies and
- Normal Forms (remember 3NF and BNCF?)
End of reminders

Goals:
✓ (logical + physical) Data independence
✓ Avoid inconsistencies

NEW PAPER - Break point!

Detailed Roadmap
- Intro
- Codd’s paper
- System R - design
- System R - evaluation
System R Architecture

- **Multiple virtual machines!**
- Programs (Sequel, QBE, etc.)
- Relational Data System (auth, integrity, view, query optimization, catalog mgmt)
- Relational Storage System (device mgmt, space alloc, buffers, Xact consistency - locking, recovery)

System R Architecture (cont.)

- Programs (Sequel, QBE, etc.)
- Relational Data Interface (called from host language, supports emulators, etc.)
- Relational Storage Interface (access to tuples)

Even more detailed Roadmap

- Intro
- Codd’s paper
- System R - design
  - RDS (QL, Data control, Q-opt)
  - RSS (Segments, rel, images, links, CC, recovery)
  - System R - evaluation
Host Language Interface

- Example:
  - EMP(EMPNO, NAME, DNO, JOB, SAL, MGR)
  - DEPT(DNO, DNAME, LOC, NEMPS)

- RDS - Embedded SEQUEL in a program:
  - CALL BIND('X', ADDR(X));
  - CALL BIND('Y', ADDR(Y));
  - CALL SEQUEL(C1, 'SELECT NAME: X, SAL: Y FROM EMP WHERE JOB="PROGRAMMER"');
  - CALL FETCH(C1);
  - CALL DESCRIBE(C1, DEGREE, P)

Gives variable address to RDI
Associate C1 to answer tuple set
Get next
Describe C1 into array

Host Language Interface (cont.)

- Locking
  - FETCH_HOLD locks
  - RELEASE unlocks

- Transaction calls (passed through to the RSI)
  - BEGIN_TRANS
  - END_TRANS
  - SAVE (checkpoint)
  - RESTORE

Queries

SEQUEL = SQL
SELECT <attribute_list> [count, avg, sum, ...] FROM <relation_list> [WHERE <condition>] [ORDER BY ...] [HAVING ...] [GROUP BY ...]
Data Manipulation

- Updates
  UPDATE <relation>
  SET <attribute = value>
  [ WHERE <condition> ]

- Insertions
- Deletions

Data Definition

- Create / Drop TABLE (=relation)
- Define / Drop VIEW (for read authorization)
  - E.g., DEFINE VIEW VEMP AS:
    SELECT *
    FROM EMP
    WHERE DNO =
    SELECT DNO
    FROM EMP
    WHERE NAME = USER;

- Expand table (add new field)

Rules

- Integrity constraints
  ASSERT ON UPDATE TO EMP:
    NEW SAL ≥ OLD SAL

- Triggers
  DEFINE TRIGGER EMPINS
  ON INSERTION OF EMP:
    (UPDATE DEPT
     SET NEMPS = NEMPS + 1
     WHERE DNO = NEW EMP.DNO)

- Catalogs (relations, views, triggers, etc.)
Optimizer

- Measure mainly I/O cost
- Emphasize importance of clustering
- Based on existence of indices
- Cost model – choose cheapest plan
- Details later...

Even more detailed Roadmap

- Intro
- Codd’s paper
- System R - design
  - RDS (QL, Data control, Q-opt)
  - RSS (Segments, rel, images, links, CC, recovery)
- System R - evaluation

RSS Segments

- Segment: logical address space
- Used to store large relations, catalogs, logs…
- No relation spans segments
- User-defined segment length
- Mapped to a set of fixed-size disk pages
  - Page map, replacement
- Segment types
  - E.g., for shared data, temporary relations, etc.
RSS log segments + recovery

- Special segment for logs
- Recovery (shadow pages)
  - Two (current and backup) page maps / segment
  - OPEN_SEGMENT: identical
  - Update request: current map to a new page
  - Replacement: send to new page
  - SAVE_SEGMENT: backup := current
  - RESTORE_SEGMENT: current := backup
- Used for checkpointing and seg. recovery

Storage System (cont.)

- Relations
  - Fixed- and variable-length attributes
  - New fields added to the right
  - Tuple id = page number + offset from bottom
  - Updates of variable-sized fields: overflow
- Links
  - Connect tuples in one (sort) or two (1:N) relations
  - Tuple=Prefix+data

Current Scheme: Slotted Pages

- How to store tuples in a page (so that tid’s remain valid)

<table>
<thead>
<tr>
<th>R</th>
<th>RID</th>
<th>SSN</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1237</td>
<td>Jane</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4322</td>
<td>John</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3563</td>
<td>Jim</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7658</td>
<td>Susan</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2534</td>
<td>Leon</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8791</td>
<td>Dan</td>
<td>37</td>
<td></td>
</tr>
</tbody>
</table>
Current Scheme: Slotted Pages

- Formal name: NSM (N-ary Storage Model)

Records are stored sequentially
- Offsets to start of each record at end of page

A Record in a Slotted Page

- All attributes of a record are stored together

Storage System (cont.)

- Images
  - … are B-tree indices
  - “Sort” relations by one or more key attributes
  - Clustered / non-clustered
  - Unique
  - Maintained by the RSS
- Links
  - Great for joins!
Concurrent Control

- Logical locking
  - Segments, relations, TIDs, key value intervals
  - Hold till end of Xact
- Physical locking (also required – why?)
  - Pages
  - Hold for a single RSI operation
- All locking is automated, and at RSS level
- 3 levels of consistency (later, later)
- Deadlock detection: youngest Xact killed

Recovery

- Needed to ensure consistency after a crash
- Checkpoints (database dumps)
- Log with old and new values
- 'soft' failure: Shadow paging
- disk failure: Logging and tape recovery

RSI Operators

- Segments
  - OPEN_SEGMENT
  - CLOSE_SEGMENT
  - SAVE_SEGMENT
  - RESTORE_SEGMENT
- Transactions/locks
  - START_TRANS
  - END_TRANS
  - SAVE_TRANS
  - RESTORE_TRANS
  - LOCK_SEGMENT
  - LOCK_RELATION
  - RELEASE_TUPLE
System R Summary

- RDS/RSS
- SEQUEL
- Transaction support
  - Concurrency control with hierarchical locks
  - Recovery with checkpoints, log and shadow paging
- Authorization/assertions/triggers
- Elaborate query optimizer
- Segments, images (indices), links

NEW PAPER - Break point!

Detailed Roadmap

- Intro
- Codd’s paper
- System R - design
- System R - evaluation
Evaluation: Goals

- High-level, data-independent Q.L.
- Support application programs & ad-hoc q’s
- Concurrency
- Recovery
- Views
- GOOD PERFORMANCE

Implementation Phases

Phase 0 [74-75]
Quick implementation: SQL subset

Phase 1 [76-77]
Implementation of full system

Phase 2 [78-79]
Evaluation

Phase 0
- XRM access method
- Single user (why?)
- SQL (mainly interactive)
  - no joins, subqueries instead
- Catalog: set of relations
  - Managed by the system like any other
  - (XRM) tuples <tid, val_ptr, val_ptr, …)
  - "inversions" (=indices)
  - Query Optimization
Lessons from Phase 0

- Materializing tuples is expensive
- CPU bound system - cost = aT_c + b (#I/O)
- Joins are important
- Query optimizer: should be geared to simpler queries

Phase 1

All of the above and...

- Compilation (R. Lorie)
  - invalid modules are recompiled transparently
  - Ad-hoc queries (UFI): same treatment
- RSS paths
  - Index scan
  - Relation scan (in physical order)
  - Link scan

Phase 1 (cont.)

- Query optimization
  - Use statistics to calculate estimates
  - Joins
    - 2-way: nested loops or sort-merge
    - N-way: tree search on 2-way combinations
Phase 1 (cont.)

- Locking
  - abandoned predicate locking (why?)

- Locking
  - abandoned predicate locking
    - slow to check conflicts; locks should be in RDS
  - Locking on physical items (hierarchies)
  - "trading" (?) and intention locks

Phase 2: Evaluation

- At IBM and customer sites for 2.5 years
- General comments
  - Enthusiastic, easy installation/reconfiguration
  - OK speed for 200Mb db, 10 conc. Users
  - slow for complex joins
Phase 2: Evaluation (cont.)

- SQL
  - Simplicity, power and data independence
  - Uniform across environments (ANSI standard)
  - User-suggested extensions (exist, like, outer join)

Phase 2: Evaluation (cont.)

- Compilation approach?
Phase 2: Evaluation (cont.)

- Compilation approach was great success
  - Short, repetitive Xacts
  - Ad-hoc queries: code generation takes little time
    - Not perceivable to the user
    - Pays off after a few records have been fetched
  - Simplified design: Same approach for all queries

Phase 2: Evaluation (cont.)

- Access paths:
  - B-trees?
  - no hashing?
  - Links?

Phase 2: Evaluation (cont.)

- Access paths:
  - B-trees,
  - no hashing,
  - no links
    - "essential": unusable by optimizer, non-nav. SQL
    - "non-essential": hard to maintain
Phase 2: Evaluation (cont.)

- Query optimizer
  - (how would you test it?)
  - (how accurate were the estimates?)

Phase 2: Evaluation (cont.)

- Query optimizer
  - Experiments on "uniform and independent" DB
  - Correct path ordering, est. costs may be off

Phase 2: Evaluation (cont.)

- Views & authorization?
Phase 2: Evaluation (cont.)

- Views & authorization: flexible & convenient

Phase 2: Evaluation (cont.)

- Recovery
  - Shadow page algo?

Phase 2: Evaluation (cont.)

- Recovery
  - Shadow page ⇒ performance penalties
    - (logging updates may be better)
Phase 2: Evaluation (cont.)

- Locking (3 levels)
  - Level 1: may read dirty data
  - Level 2: reads clean data; successive reads may give different results
  - Level 3: “Correct”
  - Q: is Level 1 faster > Level 2 > Level 3?

- Convoy phenomenon
  - Q: often, many xacts do nothing, waiting -
    - what is wrong?
    - And how to fix it?
Phase 2: Evaluation (cont.)

- Convoy phenomenon
  - Q: often, many xacts do nothing, waiting -
    - what is wrong?
    - And how to fix it?
  - A: Locks frequently requested / shortly released (like what?)
    - Solution: Round-robin CPU should NOT swap out job w/ high-traffic lock

Phase 2: Evaluation (cont.)

- Storing catalogs as relations: Good or bad?

Phase 2: Evaluation (cont.)

- Storing catalogs as relations: NICE!
  - Same QL for accessing everything
Evaluation - Conclusions

- Compilation, query optimizer
- CODASYL vs relational
  - Qopt performance worse than network model
  - But more adaptable and independent of data

Phase 2: Evaluation

- At IBM and customer sites for 2.5 years
- General comments
  - Enthusiastic, easy installation/reconfiguration
- SQL
  - Simplicity, power and data independence
  - Uniform across environments (ANSI standard)
  - User-suggested extensions (exist, like, outer join)

Phase 2: Evaluation (cont.)

- Compilation approach was great success
  - Short, repetitive Xtions
  - Ad-hoc queries: code generation takes little time
    - Not perceivable to the user
      - Pays off after a few records have been fetched
    - Simplified design: Same approach for all queries
- Access paths: B-trees, no hashing, no links
  - "essential": unusable by optimizer, non-nav. SQL
  - "non-essential": hard to maintain
Phase 2: Evaluation (cont.)

- Query optimizer
  - Experiments on "unified and independent" DB
  - Correct path ordering, est. costs may be off
- Views & authorization: flexible & convenient
- Recovery
  - Shadow page ⇒ performance penalties
    - (logging updates may be better)
- Locking (3 levels)

---

Phase 2: Evaluation (cont.)

- Convoy phenomenon
  - Locks frequently requested / shortly released
  - Round-robin CPU swaps job w/ high-traffic lock
- Storing catalogs as relations: NICE!
  - Same QL for accessing everything
- Conclusions
  - Compilation, query optimizer
  - Qopt performance worse than network
  - But more adaptable and independent of data