
System R and the Relational Model

Christos Faloutsos
www.cs.cmu.edu/~christos

Roadmap
1) Roots: System R and Ingres
2) Implementation: buffering, indexing, q-opt
3) Transactions: locking, recovery
4) Distributed DBMSs
5) Parallel DBMSs: Gamma, Alphasort
6) OO/O R DBMSs
7) Data Analysis - data mining
8) Benchmarks
9) vision statements
extras (streams/sensors, graphs, multimedia, web, fractals)

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:
- RL too much math
- Implementation
- OLTP <-> operators

Relational:
- DBTG complicated
- No easy set queries
- No semantics

Late 70’s: Relational Model wins

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

System R
- IBM SJ, 1974-78
  - compiler
  - RDS/RSS links
  - Recovery scheme
  - No hashing

INGRES
- UCB 1973-77
  - Interpreter
  - Unix FS (no recovery!)
  - 16-bit PDP-11

Impact

System R
- ES/OS / HP Allbase, IDMS/SQL,
- Oracle, DB2, SQL/DS
- Query optimization
- Compilation
- But: both systems unfaithful to Rel. Model:
  - allow duplicate records
  - No notion of domain or primary key

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

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)
- \( R (s_1, …, s_n) \subseteq S_1 \times \ldots \times S_n \)
- 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 \( \pi_{i_2} (R(s_1,s_2,s_3)) = R'(s_1,s_2) \)
- Join \( R \bowtie S \)
- Composition \( \pi_{i_3} (R \bowtie S) \)
- Restriction (selection with AND, OR)

(‘Restriction’)

\( R' = R_{\{2,3\}}^{1,2}S \)

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.

\[ \text{projection}_{\{2,3\}}(R') = \text{projection}_{\{1,2\}}(S) \]

[ 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
- ?
- ?
- ?
- ?
- ?

Today:

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

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

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.)

- 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)

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
Storage System (RSS)

- Segment: logical address space
  - Used to store large relations, catalogs, logs, …
  - No relation spans segments
  - Mapped to a set of fixed-size disk pages
    - Page map, replacement
  - Special segment for logs
  - Segment types
    - E.g., for shared data, temporary relations, etc.
  - Recovery (shadow pages)
    - Two (current and backup) page maps / segment

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>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>20</td>
</tr>
<tr>
<td>2</td>
<td>4322</td>
<td>John</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>1563</td>
<td>Jim</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>7658</td>
<td>Susan</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>2534</td>
<td>Leon</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>8791</td>
<td>Dan</td>
<td>37</td>
</tr>
</tbody>
</table>

Current Scheme: Slotted Pages

- Formal name: NSM (N-ary Storage Model)
- All attributes of a record are stored together
- Records are stored sequentially
- Offsets to start of each record are stored at end of page

<table>
<thead>
<tr>
<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>20</td>
</tr>
<tr>
<td>2</td>
<td>4322</td>
<td>John</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>1563</td>
<td>Jim</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>7658</td>
<td>Susan</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>2534</td>
<td>Leon</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>8791</td>
<td>Dan</td>
<td>37</td>
</tr>
</tbody>
</table>

A Record in a Slotted Page

- All attributes of a record are stored together
- Multi-bits: record length, etc.
- Offsets to variable-length fields

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!
Concurrency 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

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?)

Phase 1 (cont.)

- 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
  - ?

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?

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?
  - A: not that much - Level 3 is default and recommended!

Phase 2: Evaluation (cont.)

- 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

Evaluation - Conclusions

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

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