System R and the Relational Model

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

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

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

Late 70’s: Relational Model wins

Relational Prototypes
- SQL, Quel / Performance issue addressed

System R
- IBM SJ, 1974-78
- RDS/RSS links
- Recovery scheme
- No hashing
- Too many knobs
- UFI

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

System R
- ESQL/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

- Both systems: unfaithful to RL
- Duplicates
- No notion of domain or primary key

Codd, CACM’70

Goals:
- Data independence
  - Ordering (sorted vs. raw)
  - Indexing (existence or not)
- Avoid inconsistencies

The Relational Model

- Relation (dom, ... dom)
  \( R(\mathbf{s}_1, ..., \mathbf{s}_n) \subseteq S_1 \times \cdots \times S_n \)
- Rows
  - Distinct
  - Ordering doesn’t matter
- Columns
  - Order matters
  - Order + labels = unique identification
- Primary key, foreign key
First Normal Form (1NF)
- Simple domains only -> attributes
- No repeating groups
- Advantages/disadvantages?

Language
- Declaration of relations
- Queries
- Insertion/deletion/update

Operations and Rules
- Set operations on relations
- Projection $\pi_{s_1} (R(s_1, s_2, s_3)) = R'(s_1, s_2)$
- Join $R \bowtie S$
- Composition $\pi_{s_2} (R \bowtie S)$
- Restriction (selection with AND, OR)
- Redundancy (no derivable relations)
- Consistency

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.)

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

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 😊

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

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

A Record in a Slotted Page

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

Concurrency Control

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

Recovery

- Needed to ensure consistency after a crash
- Checkpoints (database dumps)
- Log with old and new values
- Shadow paging
- Logging and tape recovery
RSI Operators

<table>
<thead>
<tr>
<th>Segments</th>
<th>Transactions/locks</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN_SEGMENT</td>
<td>START_TRANS</td>
</tr>
<tr>
<td>CLOSE_SEGMENT</td>
<td>END_TRANS</td>
</tr>
<tr>
<td>SAVE_SEGMENT</td>
<td>SAVE_TRANS</td>
</tr>
<tr>
<td>RESTORE_SEGMENT</td>
<td>RESTORE_TRANS</td>
</tr>
<tr>
<td></td>
<td>LOCK_SEGMENT</td>
</tr>
<tr>
<td></td>
<td>LOCK_RELATION</td>
</tr>
<tr>
<td></td>
<td>RELEASE_TUPLE</td>
</tr>
</tbody>
</table>

System R Summary

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

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
- 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 geared to simpler queries
Phase 1

All of the above and...
- Compilation
  - Ad hoc queries are recompiled transparently
  - Ad-hoc queries: 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
- Locking
  - abandoned predicate locking
    - (slow to check conflicts; locks 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
- 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