Outline

System R discussion

Relational DBMS architecture

Alternative architectures & tradeoffs
Outline

System R discussion

Relational DBMS architecture

Alternative architectures & tradeoffs
System R Design

Already had essentially the same architecture as a modern RDBMS!

» SQL
» Many storage & access methods (B-trees, etc)
» Cost-based optimizer
» Compiling queries to assembly
» Lock manager
» Recovery via log + shadow pages
» View-based access control
System R Motivation

Navigational DBMS are hard to use

Can relational DBMS really be practical?
Navigational vs Relational Data

Why is the relational model more flexible?
Three Phases of Development

Why was System R built in 3 phases?
Storage in System R Phase 0

What was the issue with this design?

Too many I/Os:
• For each tuple, look up all its fields
• Use “inversions” to find TIDs with a given value for a field

Can also have reverse mappings (inversions)
B-tree nodes contain values of the column(s) indexed on.

Data pages can contain all fields of the record.

Give an example query that would be faster with B-Trees!
API

Mostly the same SQL language as today

Embedded SQL in PL/I and Cobol
  » .NET added LINQ in 2007

Interesting additions:
  » “EXISTS”
  » “LIKE”
  » Prepared statements
  » Outer joins

```sql
SELECT expression(s)
FROM table
WHERE EXISTS
  (SELECT expr FROM table WHERE cond)
WHERE name LIKE 'Mat%'

stmt = prepare("SELECT name FROM table WHERE id=?")
execute(stmt)
```
Query Optimizer

How did the System R optimizer change after Phase 0?
Query Compilation

Why did System R compile queries to assembly code?

How did it compile them?

Do databases still do that today?
Example 1:

```
SELECT SUPPNO, PRICE
FROM QUOTES
WHERE PARTNO = '010002'
AND MINQ<=1000 AND MAXQ>=1000;
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>CPU time (msec on 168)</th>
<th>Number of I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing</td>
<td>13.3</td>
<td>0</td>
</tr>
<tr>
<td>Access Path Selection</td>
<td>40.0</td>
<td>9</td>
</tr>
<tr>
<td>Code Generation</td>
<td>10.1</td>
<td>0</td>
</tr>
<tr>
<td>Fetch answer set (per record)</td>
<td>1.5</td>
<td>0.7</td>
</tr>
</tbody>
</table>
Recovery

**Goal:** get the database into a consistent state after a failure

“A consistent state is defined as one in which the database does not reflect any updates made by transactions which did not complete successfully.”
Recovery

Three main types of failures:
» Disk (storage media) failure
» System crash
» Transaction failure
Handling Storage Failure

DBMS

Clients

RAM

Main disk

Backup disk

Tables

Change log

Change log

(Older) tables

DBMS

CS 245
System Crash Failure

DBMS

RAM

Main disk

Backup disk

Tables

Change log

Change log

(Older) tables

Buffered pages, in-progress transactions
Handling Crash Failures: Shadow Pages

Why do we need both shadow pages and a change log?

How do shadow pages interact with disk failure?

Table

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hickey</td>
<td>House</td>
<td>123 Fantasy Way</td>
</tr>
<tr>
<td>Rat</td>
<td>Hox</td>
<td>329 Cavern Ave</td>
</tr>
<tr>
<td>Wonder</td>
<td>Woman</td>
<td>887 Truth Way</td>
</tr>
<tr>
<td>Donald</td>
<td>Duck</td>
<td>555 Quack Street</td>
</tr>
<tr>
<td>Bugs</td>
<td>Bunny</td>
<td>507 Carrot Street</td>
</tr>
<tr>
<td>Wiley</td>
<td>Coyote</td>
<td>999 Acme Way</td>
</tr>
<tr>
<td>Cat</td>
<td>Woman</td>
<td>254 Furrfect Street</td>
</tr>
<tr>
<td>Tweetie</td>
<td>Bird</td>
<td>543</td>
</tr>
</tbody>
</table>
Transaction Failure

BEGIN TRANSACTION;

SELECT balance FROM accounts
WHERE user_id = 1;

UPDATE accounts WHERE user_id = 1
SET balance = balance - 100;
COMMIT TRANSACTION;

ROLLBACK TRANSACTION;
Handling Transaction Failures

Just undo the changes they made, which we logged in the change log.

Nobody else “saw” these changes due to the locking mechanism.
Locking

The problem:
» Different transactions are concurrently trying to read & update various data records
» Each transaction wants to see a static view of the database (maybe lock whole DB)
» For efficiency, we can’t even let them do that!
Fundamental Tradeoff

Finer-grained locking

Lock smaller units of data (records or fields), lock for specific operations (e.g. R/W)

+ Allows more transactions to run concurrently
– More runtime overhead

Coarser-grained locking

Lock bigger units of data (e.g. whole table) for broader purposes (e.g. all operations)

+ More efficient to implement
– Less concurrency

Sometimes, even fine-grained locking would not be enough even if it were free!
Fundamental Tradeoff

Strong isolation level

Finer-grained locking
- Lock smaller units of data (records or fields), lock for specific operations (e.g. R/W)
  + Allows more transactions to run concurrently
  - More runtime overhead

Coarser-grained locking
- Lock bigger units of data (e.g. whole table) for broader purposes (e.g. all operations)
  + More efficient to implement
  – Less concurrency

Weak isolation level

Closer to exclusive view of DB (can’t see others’ changes)

See others’ changes, but more concurrency
Locking and Isolation in System R

Locking:
» Started with “predicate locks” based on expressions: too expensive
» Moved to hierarchical locks: record/page/table, with read/write types and intentions

Isolation levels:
» Level 1: Transaction may read (but not update) uncommitted data; successive reads to a record may return different values
» Level 2: Transaction may only read committed data, but successive reads can differ
» Level 3: Successive reads return same value

Most apps chose Level 3 since others weren’t much faster
Are There Other Ways than Locking to do Concurrency?
Authorization

**Goal:** give some users access to just parts of the database

» A manager can only see and update salaries of her employees
» Analysts can see user IDs but not names
» US users can’t see data in Europe
Authorization

System R used view-based access control

- Define SQL views (queries) for what the user can see and grant access on those

```
CREATE VIEW canadian_customers AS
SELECT customer_name, email_address
FROM customers
WHERE country = "Canada";
```

Elegant implementation: add the user’s SQL query on top of the view’s SQL query
User Evaluation

How did the developers evaluate System R?

What was the user feedback?