Database Systems
A History and Evaluation of System R

Goals

- ❑ To provide a high-level, nonnavigational user interface for maximum user productivity and data independence.
- ❑ To support different types of database use including programmed transactions, ad hoc queries, and report generation.
- ❑ To support a rapidly changing database environment, in which tables, indexes, views, transactions, and other objects could easily be added to and removed from the database without stopping the system.
- ❑ To support a population of many concurrent users, with mechanisms to protect the integrity of the database in a concurrent-update environment.
- ❑ To provide a means of recovering the contents of the database to a consistent state after a failure of hardware or software.
- ❑ To provide a flexible mechanism whereby different views of stored data can be defined and various users can be authorized to query and update these views.
- ❑ To support all of the above functions with a level of performance comparable to existing lower-function database systems.

Key Features and basic idea of key contributions

System R was designed and developed over a period of 1974-79 at IBM San Jose Research Center. It is a prototype and its purpose was to demonstrate that it is possible to build a Relational System that can be used in a real life environment to solve real life problems, with performance at least comparable to that of existing system. Its two subsystems are Research Storage and System Relational Data System.

The History of System R can be divided into three phases

Phase Zero:

- Phase Zero involved development of SQL interface and a quick implementation of a subset of SQL for one user at a time. It provided valuable insight in several areas, but its code was eventually abandoned. It uses relational access method called XRM. Since XRM is a single-user

access method without locking or recovery capabilities, issues relating to concurrency and recovery were excluded. In it interpreter program was written in PL/I to execute statements in high-level SQL. SQL includes queries and updates of database as well as creation of new relations. Implementation contained “subquery” construct of SQL but not “join” construct. It was intended for use as standalone query interface. System Catalog was stored as a regular set of relations in the database itself. Phase zero was strongly influenced by the facilities of XRM.

- XRM Storage Structure.
    - Stores relations as tuples. Tuple contains pointers to the domains and each domain may have an “inversion” (associated domain values with TIDs).

- The most challenging task in Phase Zero was the design of optimizer algorithms for efficient execution of SQL and the objective was to minimize the number of tuples fetched from the database in processing a query. Therefore, made extensive use of inversions and often manipulated TID lists.

Results of Phase zero:

It was a good idea to plan to throw away the first implementation. It demonstrated usability of SQL language.

Feasibility of new tables and inversions “on the fly” and relying on an automatic optimizer for access path selection.
Convenience of storing the system catalog in the database itself.

Lessons from Phase zero:

The optimizer should take into account not just the cost of fetching tuples, but the costs of creating and manipulating TID lists.
“Number of I/Os” is a better cost of measure than “Number of tuples fetched”.
Optimizer cost measure should be a weighted sum of CPU time and I/O count, with weights adjustable according to system configuration.
“join” formulation of SQL is very important.
The Phase Zero optimizer was quite complex and was oriented towards complex queries.

Phase One:

Involved design and construction of full function, multiuser version of System R. System R consisted of an access Method : Research Storage System (RSS) and an optimizing SQL processor: Relational Data System (RDS). RDS runs on top of RSS.

RSS does locking and logging. It does authorization and access path selection and was designed to support multiple concurrent users. Supports both PL/I and COBOL

Compilation Approach:

It is possible to compile very high-level SQL statements into compact efficient routines in System/370 machine language and SQL statements of arbitrary complexity can be decomposed into a relatively small collection of machine language “fragments”. And then an optimizing compiler can assemble these to process a given SQL statement.

Access module performs all interactions with the database by means of calls to the RSS.

Overhead of parsing, validity checking and access path selection is removed from executing program and is done in a separate preprocessor step.

Possibility that subsequent changes in database may invalidate some decisions in an access module.
Dependencies on database objects (tables, indexes) are recorded for each access module in system catalog.
If the structures invalidates an access module, it is regenerated from its original SQL statements.
Ad hoc queries coming from UFI are also converted to machine-language routines, which are executed the same way as access modules.

RSS Access Paths:

RSS stores data values in individual records and records become variable in length and longer on the average than XRM records. All data of a record is fetched in single I/O.
In place of “inversions” RSS provides “indexes” implemented in form of B-Trees. RSS also implements “links”

Link scans (from record to record)

- Search arguments can be specified, which limit the number of records returned.
- RSS also provides a built in sorting mechanism, which can sort scan results.

The Optimizer:

Designed to minimize the weighted sum of the predicted number of I/Os and RSS calls in processing an SQL statement. It uses indexes instead of TID lists.

The access path choice is based on the optimizers estimate of both the clustering and selectivity properties of each index.

Technique of performing joins originate from a research made on 10 methods. Nearly optimal 2 methods were:

- Scan over the qualifying rows of tableA, for each row, fetch the matching rows of table B.
- Sort the qualifying rows of Tables A andB in order by their respective join fields. The scan over the sorted lists and merge them by matching values

Views and Authorization:

Objective: power and flexibility.

Any SQL query to be used as definition of a view. View definitions stored in form of SQL parse trees. Operation parse tree merged with view parse tree, when an SQL operation is to be executed against a view. View can be updated only if it is derived from a single table in the database. Based on privileges controlled by the SQL statements GRANT and REVOKE. Each user can be given RESOURCE privilege, which enables him to create new tables in DB. Creator receives access,update and destroy privileges on that table. The creator can then grant these privileges to other people and each granted privilege may optionally carry with it the “GRANT” privilege. REVOKE destroys whole chain of granted privileges.

Recovery:

Objective: provision of a means whereby the database may be recovered to a consistent state in the event of a failure.

Media failure: information on disk is lost mage dump of the database plus a log of “before” and “after” changes provide the alternate copy which makes recovery possible.

Use of “dual logs” even permits recovery from media failures on the log itself.

System failure: information in main memory is lost. System R uses change log plus “shadow pages” to recover from system failure.

Transaction failure: all changes made by the failing transaction must be undone. System R simply processes the change log backwards to remove all changes made by failed transaction.
Unlike media and system recovery, which both require that System R be reinitialized, transaction recovery takes place on-line.

Linking

The original design involved concept of “predicate locks” in which the lockable unit was a database property such as “employees whose location is Evanston”

- Because it was difficult to determine that two predicates are mutually satisfiable and was also time-consuming. Two predicates may appear to conflict, when in fact the semantics of the data prevent any conflict. Desire to contain locking subsystem entirely within RSS thus make it independent of any understanding of predicates.

The chosen scheme involves a hierarchy of locks, with several sizes of lockable units, ranging from individual records to several tables.

Locking subsystem is transparent to end-users, but acquires locks on physical objects in the database as they are processed.

When a user accumulates many small locks, they can be traded for a larger lockable unit.

When locks are acquired on small objects, “intention” locks are simultaneously acquired on the larger objects which contain them.

Phase Two:

Evaluation phase lasted 2.5 years Experiments performed on the system at the San Jose Research Laboratory.
Actual use of the system at a number of internal IBM sites and at three selected customer sites.
At all user sites, System R was installed on an experimetal basis for study purposes only.

General User Comments:

System performance tuneable without impacting end users.
Performance characteristics and resource consumption generally satisfactory.
In general databases were smaller than one 3330 disk pack (200Mb) and were typically accessed by fewer than ten concurrent users.
Interactive response slowed down during execution of complex SQL statements involving joins of several tables

The SQL Language

1. Successful in achieving its goals of simplicity, power and data independence
2. Users without prior experience were able to to learn a usable subset on their first sitting.
3. As a whole the language provided the query power of the first order predicate calculus combined with operators for grouping arithmetic and built-in functions such as SUM and AVERAGE.

4. Users praised the uniformity of the SQL syntax across the environments of application programs, ad hoc query, and data definition.

Implemented User Suggestions

a. Easy to use syntax for existence or non existence of data item: “EXISTS”.
b. Searching for partially known strings: “LIKE”.
c. Requirement for computing and SQL statement dynamically, submit statement to optimizer for access path selection, then execute the statement repeatedly for different data values without re invoking the optimizer: “PREPARE” and “EXECUTE” statements in host-language version of SQL.
d. Need for “outer join” facility for SQL.

The Compilation Approach

1. The approach of compiling SQL statements into machine code was one of the most successful parts of the project
2. A machine language routine was generated to execute any SQL statement of arbitrary complexity by selecting code fragments from a library of approximately 100 fragments.
3. For short, repetitive transactions, the benefits are obvious: most overhead is removed.
4. In ad hoc query environment the advantages of compilation are less obvious as query is executed only once.
5. Final advantage is its simplifying effect on system architecture. (ad hoc queries and pre canned transactions being treated the same way).

Available Access Paths

1. The principal access path used for retrieving data associatively by its value is B-Tree index.
2. Hashing and direct links techniques were not used. They would have enhanced the performance of “canned transactions” which only access a few records.
3. For transactions which retrieve a large set of records, the additional I/Os caused by indexes are less important.

The Optimizer

A series of experiments were conducted to evaluate the success of System R optimizer. Optimizer was modified to generate every possible access path, and to estimate cost of each path. A Mechanism was added to force execution of an SQL statement by a particular access path and measure actual number of page fetches and RSS calls. Although optimizer was able to correctly order the access paths, magnitudes of predicted costs differed from measured costs in several cases.

1. Cause: inability to predict how much data would remain in system buffers during sorting.
2. The experiments conducted do not address the issue, whether or not a very good access path for a given SQL statement might be overlooked.

Views and Authorization

Users generally found Mechanisms for defining views and controlling authorization to be powerful, flexible and convenient.

Beneficial features:

1. Full query power of SQL is made available for defining new views.
2. The authorization system allows each installation of System R to choose “fully centralized”, “fully decentralized” or an intermediate policy.

Following suggestions were made to improve:

i. Authorization subsystem could be augmented by the concept of a “group” of users.
ii. A new command could be added to SQL language to change the ownership of a table from one user to another..
iii. Occasionally it is necessary to reload an existing table in the database (e.g to change its physical clustering properties) While doing this views and authorizations defined on the table are lost) It was suggested that views and authorizations be held “in abeyance” pending reactivation of the table.

The Recovery Subsystem

The combined “Shadow page” and log mechanism used in System R proved to be quite successful.

Keeping of shadow pages for each updated page had a big impact on system performance due to the primarily following factors:

- Each updated page is written to a new location on disk, the ability of the system to cluster related pages in secondary storage to minimize disk arm movement is limited.
- Since each page can be an “old” and “new” version, a directory must be maintained to locate each version
- The periodic checkpoints which exchange the “old” and “new” pointers generate I/O activity and consume certain amount of CPU time.
- Possible alternative is to dispense with the concept of shadow pages and simply keep a log of all database updates.
- Mechanisms can be developed to minimize I/Os by retaining updated pages in the buffers until several pages are written out at once, sharing an I/O to the log.

The Locking Subsystem

a. The locking subsystem provides each user with a choice of three levels of isolation from other users.

b. Under no circumstances can a transaction at any isolation level, perform updates on the uncommitted data of another transaction.
c. Level 1: may read but not update uncommitted data.
d. Level 2: transaction is protected from reading uncommitted data.
e. Level 3: Transaction is guaranteed that successive reads of the same record will yield same value.