Business System 12

An industrial-strength RDBMS (where R means R!) made by IBM, 1978-82.

Hugh Darwen
HD@TheThirdManifesto.com

for the
TTM Implementers' Workshop

University of Northumbria, 2-3 June, 2011

A Brief History of Data

1960: Punched cards and magnetic tapes
1965: Disks and 'direct access'
1970: E.F. Codd's great vision:
"A Relational Model of Data
for Large Shared Data Banks"
1970: C.J. Date starts to spread the word
1975: Relational Prototypes in IBM:
PRTV (ISBL), System R (SQL)
1980: First SQL products: Oracle, SQL/DS
1986: SQL an international standard
1990: OODB – didn't come to much in the end
2000: XML? (shudder!)

2

A Brief History of Me

1967 : IBM Service Bureau, Birmingham (UK)
1969 : "Terminal Business System" – putting users in direct contact with their databases.
Development started in USA, early '60s, then at international centre in London.
1972 : Attended Date's course on database (a personal watershed)
1978 : "Business System 12"
- a relational dbms for the Bureau Service
(International centre moved to The Netherlands & some Brits went with it)
1985 : Death of Bureau Service (and of BS12)
1987 : Joined IBM Warwick dev. lab. Attended a Codd & Date database conference in December, leading to:
- start of collaboration with CJD
- start of participation in SQL international standard
2004 : Retired (to teach relational db theory)

3

Terminal Business System

- A multi-user DBMS before the term was coined
- Prerelational but ahead of its time in many respects
- Very strong on security, recovery, transactions
- Proprietary record-level access methods using keys (and under-the-covers hashing)
- Included three general-purpose "utilities", via a rudimentary DDL called RAF (Record And File descriptors):
  - "File Maintenance": record-level add/change/delete by key value, using field name/value pairs.
  - Single record "Inquiry" by file name/key value, names of required fields.
  - Comprehensive "Report Writer" — very popular but customers wanted more than we could provide …

4

Report Writer Deficiencies

- Single input file
- Single record format per file
- No pointer-chasing

In 1972 HD given task of devising solutions to overcome these restrictions.

In same year, HD sent on CJD's course on "databases". Introduced to Codd's RM.

5

How The RM Could Come to The Rescue

- Single input file: replace by relational query!
- Single record format per file: no longer a problem
- No pointer-chasing: no longer a problem

But then we would have to start with a blank sheet: build a brand new DBMS

6

Planning Business System 12

In 1978. Questions arising from study of Codd's papers:

1. What about "calculations"?

   Answer provided by both ISBL and SQL (System R): EXTEND and SUMMARIZE in **Tutorial D**

2. How to handle duplicate attribute names arising from, e.g., equijoin? (Dot qualification clearly unsatisfactory)

   Answer provided by ISBL (and not by SQL!)

3. What does he really mean by "domain"?

   Answer provided by nobody (at that time), and we got it badly wrong. (The SQL standard made the same mistake 14 years later!)

7

Some General Language Features

- No semicolons: statements separated by line breaks unless line ends in + (ignore leading blanks on next line) or – (don't ignore)
- Nearly all key words abbreviatable down to 1st 3 chars (e.g., INT[ersect], JOI[n], SQR[t]). Many reserved words.
- All proprietary and user-defined operators invoked in prefix notation. But infix for usual arithmetic operators.
- English key words for commands, prefix operators and options (also using prefix notation)

  e.g. STORE result, INTO(snapshot), REPLACE (relational assignment)
  SELECT(SP, QTY > 50) a relational operator invocation

8

Notable Features in Release 1

We didn't wait for our first users to ask for these:

- User-defined operators: procs & funcs
- "Call Level Interface": hides burdensome client-server communication protocol (proprietary) from apps
- Deferrable constraint checking sort of, via BYPASS option (authorisation needed!) and VALIDATE *table-name* command.
- Nested transactions via START/END/CANCEL UNIT | ALL | *name*
- Temporary tables: Local to session, destroyed on session end.
- Triggered procedures
- CREATE … LIKE(*table-exp*)
- system-generated column values

9

Scalar Types (= "System Domains")

Built-in types only (but see domains), all except SYNtax 1st class):

- CHAracter as in **Tutorial D** (no length restriction)
- NUMeric (no scale or precision restrictions)
- TIMestamp (no separate DATE type!)
- BIT for "truth" values, denoted by literals '0'B (false) and '1'B (true)
- NAMe used for identifiers (table names etc.), case-insensitive, blank-separated word strings: LENGTH(*n*) gives no. of words.
- SYNtax for local vars & parameters only, values denote chunks of BS12 syntax. All procedure parameters are of this type because arguments are substituted, not evaluated.

10

Non-Scalar Types

- TABLE types only (no row types, no row expressions)
- The only types available for database variables
- Available as parameter types for functions and "views" …
- … but not available for columns or local variables
- Zero or more columns, with unique names, not considered to be in any order
- Columnless table types available for intermediate results in table expressions only, alas!

11

"The 12" Relational Operators

- GENERATE (*assign-list*): selector for 1-row tables
- CALCULATE(*t, assign-list*): = extension
- SUMMARY(*t,* GROUP(*col-list*), *agg-assign-list*):
  = SUMMARIZE … BY … in **Tutorial D**
- SELECT(*t, condition*) = restriction
- PRESENT(*t, see next slide*) = projection with renaming
- JOIN(*t1, t2, …*) = *n*-adic natural join
- QUAD(*t1, t2, …*) = *n*-adic Cartesian product (no common cols)
- INTERSECT(*t1, t2*) = *t1* MATCHING *t2* in **Tutorial D**
- DIFFERENCE(*t1, t2*) = *t1* NOT MATCHING *t2* in **Tutorial D**
- UNION(*t1, t2, …*) = *n*-adic union of projections on common attributes
- EXCLUSION(*t1, t2*) = UNI(*t1* DIF *t2* , *t2* DIF *t1*)
- MERGE(*t1*,*t2*[,*t3*]) = *n*:1 dyadic join, "outer" if 1-row table *t3* given

big mistake!

12

BS12's Projection/Rename Operator

PRESENT (*table, column-spec-commalist*)
where a *column-spec* can be:
ALL
NONE
[INCLUDE(] *column-name-list* [)]
EXCLUDE(*column-name-list*)
RENAME(*column-name1, column-name2*)
and a *column-name-list* can be:
a commalist of column names, or
LIST(*t, c*) such that the result of PRESENT(*t,c*) provides the column names, where *c* is the name of a column in *t* of type NAME

13

Other Uses of LIST(*t*,*c*)

UNION | JOIN | QUAD (LIST(*t,c*))

where *c* is a column of type CHAR in table *t* and the list of operand table-expressions is derived from PRESENT(*t,c*).

Can be used to express queries such as "In which columns of which *stored tables* does the value *v* appear?"

i.e., base relvars

14

Other Operators of Type TABLE

These are all to do with "system" or "catalog" information.
Built-in function invocations were used rather than table names.

- COLUMNS (*table-exp*)
- TABLES(OWN | SESSION | INSTALL | SYSTEM | *userid*)
- DOMAINS(OWN | INSTALL | SYSTEM)
- ACCESS – who can access which of my tables?
- ACCESS(*userid*) – which of *userid*'s tables can I access?
- SYNONYMS and SYNONYMS(INSTALL)
- TEMP and TEMP(*temporary view def name*) ← See next slide
- SYSTEM(PROFILE)
- SYSTEM(MESSAGES) from system or other users
- HISTORY(BACKUP) and CONTENTS(BACKUP)
- HISTORY(RESTORE) and REQUESTS(RESTORE)

… and several more but I ran out of room

15

The DEFINE Command

[DEFine] *introduced-name* | * = *table-expression*

a very special "name"

e.g. t1 = SEL(S,CITY = 'Paris')
t2 = JOI(t1, SP)
ans = PRE(t2, EXC(CITY))

t1, t2, and ans are "temporary view definition names"

KEEP command available to make them permanent
e.g. KEE ans
or KEE ans AS(Parts supplied by Parisians), REPLACE

Many thanks to ISBL for DEF and KEE!

16

Updating

Syntactically, any table-exp can be an update target.

Single row updates:
ADD | CHANGE | DELETE *t, key-col-spec-list* [*, assign-list*]
E.g. ADD SP, SNO('S1'), PNO('P4'), QTY = 1000
CHA S, SNO('S1'), STATUS = 40
DEL P, PNO('P2')

Multi-row updates (nowadays called insert, update, delete)
STORE *t1,* [INTO(] *t2* [)] [, *option-list*]
UPDATE *t1* [, USING(*t2*)] , *assign-list* [, *option-list*]
CLEAR *t1*
PROCESS *user-defined-proc*(*args*), *t* [*, option-list*]

Options: ERROR(CONTINUE | END | CANCEL)
BYPASS certain constraint checks (a bit murky, this!)
ORDER(*ordering-spec*) for USING and PROC

17

Updating through *table-exp*

The following are updatable if operands are updatable:

- PRESENT (*t*, …) so long as the key of *t* is preserved
- SELECT (*t, cond*) so long as *cond* is satisfied
- CALCULATE (*t, assign-list*) original cols only
- INTERSECT (*t1, t2*) so long as *t1* rows stay "matching"
- DIFFERENCE (*t1, t2*) so long as *t1* rows stay "not matching"
- MERGE (*t1, t2*) but not JOIN (*t1, t2,* …)
- TABLES(OWN) though CREATE shorthand is available
- COLUMNS(*t1*) (CREATE does inserts into this)
- DEFINITION(*t1*) if *t1* a language table
- DOMAINS(OWN) but existing data not rechecked! (VALIDATE *t* available for this)

No shorthand available for updating DOMAINS(OWN)

18

Integrity

Pretty good for its time but pretty dreadful by *TTM* standards:

- Every table (not just stored tables) has exactly one key
- Every column is defined on a domain, giving:
  - data type (a.k.a. "system domain")
  - max length (0=unlimited or inapplicable)
  - "auto gen" option: serial or timestamp
  - scale for numbers (as positions "after" the point, can be -ve)
  - default formatting for data transfer to and from client
  - default value (an expression)
  - check expression (can access database, as in SQL)
  - units (for info only, "e.g. Dutch guilders" says the book!)
  - comments (for info only)

But domain check expressions can be BYPASSed, if user authorised, even across transaction boundaries (it seems).

19

Keys

Exactly one must be specified for every stored table.
Some crude heuristics used to work out keys for table-exps:

- PRESENT (*t*, …) key of *t* if preserved, else all-key
- SELECT (*t, cond*) key of *t*
- CALCULATE (*t, assign-list*) key of *t* if preserved, else all-key
- INTERSECT (*t1, t2*) key of *t1*
- DIFFERENCE (*t1, t2*) key of *t1*
- MERGE (*t1, t2* …) key of *t1*
- QUAD (*t1, t2*) union of keys of *t1* and *t2*
- JOIN (*t1, t2*) if *m*:1, key of "*m*" operand, else union of keys
- UNION (*t1, t2*) all-key
- SUMMARY (*t,* GROUP(*g*), …) *g* is the key
- EXCLUSION (*t1, t2*) can't remember (all-key?)
- GEN (…) all-key?--I hadn't yet discovered the empty key!

20

CREATE Options

- CONTENT (DATA | CLIST | PROCESS | FUNCTION | VIEW)
  if not DATA, then it's a "language table", cols LINENO (key) & TEXT
- LIKE (*table-exp*)
- SESSION
- REPLACE
- KEY (*column-spec-commalist*) for key cols if SERIAL not used
- COLUMNS (*column-spec-commalist*) for nonkey cols
- SERIAL (*column-name*) for auto-gen serial key
- ORDER (*order-spec*) for default order on xfer to client

After CREATE *t*, updates on COLUMNS(*t*) are available but restrictions apply as soon as *t* becomes nonempty for the first time.

21

Language Tables

- Special stored tables for procedures, functions, views
- Purpose indicated by CONTENT value in TABLES row
- Code denoted by DEFinition(*table-name*)
- Columns LINENO (key) and TEXT
- Subject to regular update commands
  (but front-ends provided proper editors)
- KEEP *v* generates line numbers and preserves all the DEFINE invocations involved in the definition of *v*

22

Data Transfer

For data transfer to or from client application:

START TRANSFER *table-exp* [, *options*]

Options include:

- GET [, MODIFY] = intent to retrieve rows and possibly update them in "where current" fashion (blush!)
- PUT [, REPLACE | APPEND] = intent to add rows (REPLACE = delete all existing rows first)
- UPDATE = intent to use ADD, CHANGE, DELETE operations on *table-exp*
- ORDER (*order-spec*) (for use with GET)
- FORMAT (*colname-1*(*format-spec-1*), …, *colname-n*(*format-spec-n*))

Bit like cursors in SQL, but no more than one open at a time.
Format-specs were *very important*. E.g., allowed numeric values to be presented in packed decimal, floating point, 32-bit binary etc. 30 pages on this in the reference manual!

23

Implementation

24

Time Line and People

1978: Technical planning and specification in The Netherlands. About 5 people. HD devises gross architecture.

1979-81: Development of Release 1 in the Netherlands
About 15 programmers? (I can remember 11 by name)
HD responsible for execution code for relational stuff

1982: HD and other Brits return to UK while Release 2 development (row-level locking) proceeds in The Netherlands

1982-83: HD and others work on additional client-side stuff
- A "call-level interface" lookalike (cf. SQL/CLI or ODBC)
- HD's "baby": Dbird, all-singing-and-dancing, menu-driven, highly acclaimed, award-winning general purpose application *including the report generator he had dreamed of in 1972.*

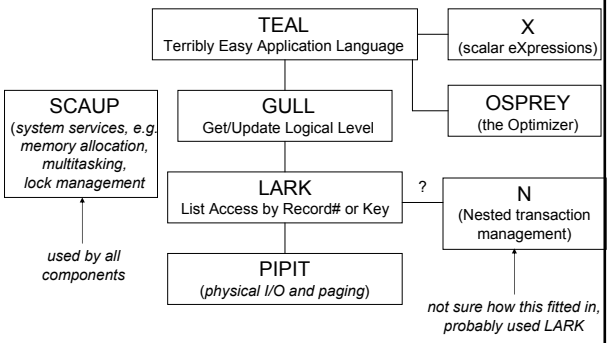
25

Environment

- Server (DBMS) runs as single "job" in OS/MVS on S/370
  - proprietary protocol for communication with clients
  - basic, page-level, VSAM files for databases: each user has one for permanent store, one for "scratchpad" (session tables and intermediate results spun off during query evaluation)
- Client-independent in principle but expected clients run in VSPC, our proprietary replacement for TSO, on same machine
  (support for clients at remote PCs was later developed for the UK service)

26

Gross Architecture



27

The Pipelining Tree

- root node: final result (queries) or target (updates)
- intermediate nodes: relational operators plus "go faster" and "duprem" nodes inserted by optimizer
- leaf nodes: stored tables and GENERATE invocations
- common interface for all nodes (determine key, prepare, identify next row, reset, add, change, delete, etc.)
  Each node type provided its own implementation for each op
  Entry point addresses for each op stored in node control block (*this was before the advent of OO, remember!*)
  So, e.g., leaf nodes for stored tables used GULL

28

The Optimizer

- cost-based, using crude cardinality estimates
- tree rearrangement and extension:
  - for join-like ops, determine whether m:n, m:1, or 1:1 (based on knowledge of keys)
  - switch join operands where appropriate (e.g., "1" side goes to the right)
  - move selects below joins etc.
  - detect need for duplicate removals and insert duprem nodes in "best" places
  - insert go-faster nodes, e.g., sorts, direct access by key value at leaf

One big mistake: no interface between OSPREY and X.
So optimizer couldn't analyse scalar expressions for, e.g., "key equal" comparisons. Consequence: JOIN(S, GEN(SNO='S1')) went much faster than SELECT(S, SNO='S1') !!
I believe they did something about this in Release 2 (there was no Release 3)

29

Optimizing Duprem

- Perhaps the single most crucial implementation feature
  (slavish adherence to the RM, strongly advocated by HD, had been somewhat controversial)
- "Fussy nodes" requiring duprem:
  - final result for queries (but not for invocations of EMPTY)
  - SUMMARY
  and no others!
- "Rogues", giving rise to dups:
  - projection (PRESENT) when key not preserved
  - UNION
  - extension (CALCULATE) when key column replaced
  and no others (no RVA support in BS12, so no UNGROUP)
- Insert duprem node at "best" place between rogue and fussy node

30

LARK Features

- Supports storage structures called lists:
  - for the whole database, including the "data dictionary" (= catalog nowadays)
  - fixed-length elements
  - sequential lists, elements accessed by record number
  - hash lists, elements accessed by key value using "dynamic extendible hashing" technique that had only just been published by Ron Fagin of IBM (*amazingly timely publication – TBS's hashing required preallocated nonextendible storage and we knew this wouldn't be good enough for BS12)*
  - B-trees added in Release 2 (after HD's departure)
- Only user of PIPIT (for page allocation and disk I/O)
- GULL, for stored tables, (and N, for logs?) its only user(s)

31

GULL Features

- In charge of stored tables:
  - vertical decomposition to avoid oversized list elements (via ADD operation at leaf node for TABLES(OWN)!)
  - hashed "literal pool" for character strings longer than 12 (split into chained chunks)
  - hash lists for key columns (except "auto-gen serial" keys, which just become record numbers and take no space at all)
  - sequential lists for non-key columns
  - record numbers and "pseudokeys" (hash tokens) provide connections between list elements belonging to same stored row
- "Identify" positions cursor on a stored row but does not read data.
- Subsequent "realise" obtains specified column values
  so can be delayed until absolutely needed (if at all)
  But this technique was not as successful, performance-wise, as we had hoped.

32

Lots More Could Be Said, but this is

The End

33