CHAPTER 4: RELATIONAL DATABASE SYSTEM

Introduction to relational database management system

RDBMS (relational database management system), A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organised in tables.

A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables

Characteristics of Relational Databases

CODD'S RULES FOR RELATIONAL DATABASES, The relational model for databases described by Dr. Codd contains 12 rules.

Relational Database Characteristics

1) Data in the relational database must be represented in tables, with values in columns within rows.
2) Data within a column must be accessible by specifying the table name, the column name, and the value of the primary key of the row.
3) The DBMS must support missing and inapplicable information in a systematic way, distinct from regular values and independent of data type.
4) The DBMS must support an active on-line catalogue.
5) The DBMS must support at least one language that can be used independently and from within programs, and supports data definition operations, data manipulation, constraints, and transaction management.
6) Views must be updatable by the system.
7) The DBMS must support insert, update, and delete operations on sets.
8) The DBMS must support logical data independence.
9) The DBMS must support physical data independence.
10) Integrity constraints must be stored within the catalogue, separate from the application.
11) The DBMS must support distribution independence. The existing application should run when the existing data is redistributed or when the DBMS is redistributed.
12) If the DBMS provides a low level interface (row at a time), that interface cannot bypass the integrity constraints.

Relational algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yields relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

Fundamental operations of Relational algebra:

- ✓ Select
- ✓ Project
- ✓ Union
- ✓ Set different
- ✓ Cartesian product
- ✓ Rename

These are defined briefly as follows:

Select Operation (σ)

Selects tuples that satisfy the given predicate from a relation.

Notation σ*p*(r)

Where *p* stands for selection predicate and r stands for relation. *p* is prepositional logic formulae which may use connectors like and, or and not. These terms may use relational operators like: =, ≠, ≥, < , >, ≤.

For example:

σ*subject="database"*(**Books**)

Output : Selects tuples from books where subject is 'database'.

σsubject="database" and price="450"(Books)

Output: Selects tuples from books where subject is 'database' and 'price' is 450.

σsubject="database" and price < "450" or year > "2010"(Books)

Output : Selects tuples from books where subject is 'database' and 'price' is 450 or the publication year is greater than 2010, that is published after 2010.

Project Operation (Π)

Projects column(s) that satisfy given predicate.

Notation: ΠA1, A2, An (r)

Where a1, a2 , an are attribute names of relation r.

Duplicate rows are automatically eliminated, as relation is a set.

for example:

$\Pi_{subject, author}$ (Books)

Selects and projects columns named as subject and author from relation Books.

Union Operation (∪)

Union operation performs binary union between two given relations and is defined as:

$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

Notion: r U s

Where r and s are either database relations or relation result set (temporary relation).
For a union operation to be valid, the following conditions must hold:

- r, s must have same number of attributes.
- Attribute domains must be compatible.

Duplicate tuples are automatically eliminated.

Π author (Books) $\cup$ Π author (Articles)

Output: Projects the name of author who has either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples which present in one relation but are not in the second relation.

Notation: r − s

Finds all tuples that are present in r but not s.

Π author (Books) − Π author (Articles)

Output: Results the name of authors who has written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation: r X s

Where r and s are relations and there output will be defined as:

$r X s = \{ q t \mid q \in r \text{ and } t \in s\}$ $\Pi_{\text{author = 'tutorialspoint'}}$(Books X Articles)

Output : yields a relation as result which shows all books and articles written by tutorialspoint.

Rename operation (ρ)

Results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. rename operation is denoted with small greek letter rho *ρ*
Notation: *ρ* x (E)
Where the result of expression E is saved with name of x.
Additional operations are:

- Set intersection
- Assignment

Natural join

Relational Algebra Types and Operations

An algebra is a formal structure consisting of *sets* and *operations* on those sets.

Relational algebra is a formal system for manipulating relations.

- Operands of this algebra are relations.
- Operations of this algebra include the usual set operations (since relations are sets of tuples), and special operations defined for relations
    - selection
    - projection
    - join

Relational algebra is a formal system for manipulating relations (Tables). Operands of this algebra are relations. Operations of this algebra include the usual set operations (since relations are sets of tuples [Rows]), and special operations defined for relations.

Set Operations on Relations

For the set operations on relations, both operands must have the same scheme, and the result has that same scheme.

- R1 U R2 (union) is the relation containing all tuples that appear in R1, R2, or both.
- R1 n R2 (intersection) is the relation containing all tuples that appear in both R1 and R2.
- R1 - R2 (set difference) is the relation containing all tuples of R1 that do not appear in R2.

Selection (σ)

Selects tuples from a relation whose attributes meet the selection criteria, which is normally expressed as a predicate.

```
R2 = select(R1,P)
```

That is, from R1 we create a new relation R2 containing those tuples from R1 that satisfy (make true) the predicate P.

A *predicate* is a boolean expression whose operators are the logical connectives (and, or, not) and arithmetic comparisons (LT, LE, GT, GE, EQ, NE), and whose operands are either domain names or domain constants.

select(Workstation,Room=633) =

```
Name     Room     Mem    Proc  Monitor
==================================
coke      633   16384  SP4   color17
bass      633    8124  SP2   color19
bashful   633    8124  SP1   b/w
```

select(User,Status=UG and Idle<1:00) =

```
Login    Name         Status   Idle  Shell  Sever
=============================================
jli      J. Inka      UG       0:00  bsh    UG
```

Projection (Π)

Chooses a subset of the columns in a relation, and discards the rest.

```
R2 = project(R1,D1,D2,...Dn)
```

That is, from the tuples in R1 we create a new relation R2 containing only the domains D1,D2,..Dn.

project(Server,Name,Status) =

```
Name     Status
==============
diamond   up
emerald   up
graphite  down
ruby      up
frito     up
```

```
project(select(User,Status=UG),Name,Status) =

Name         Status
==================
A. Cohn        UG
J. Inka        UG
R. Kemp        UG
```

Join

Combines attributes of two relations into one.

```
R3 = join(R1,D1,R2,D2)
```

Given a domain from each relation, *join* considers all possible pairs of tuples from the two relations, and if their values for the chosen domains are equal, it adds a tuple to the result containing all the attributes of both tuples (discarding the duplicate domain D2).

Natural join: If the two relations being joined have exactly one attribute (domain) name in common, then we assume that the single attribute in common is the one being compared to see if a new tuple will be inserted in the result.

Assuming that we've augmented the domain names in our lab database so that we use MachineName, PrinterName, ServerName, and UserName in place of the generic domain "Name", then

```
join(Workstations,Printers)
```

is a natural join, on the shared attribute name Room. The result is a relation of all workstation/printer attribute pairs that are in the same room.

Example Use of Project and Join

Find all workstations in a room with a printer.

- R1 = project(Workstation,Name,Room)
- R2 = project(Printer,Name,Room)
- R3 = join(R1,R2)

```
R1              R2              R3
Name    Room    Name    Room    WName   Pname   Room
==========      ==========      ====================
coke     633    chaucer  737    coke    uglab   633
bass     633    keats    706    bass    uglab   633
bashful  633    poe      707    bashful uglab   633
tab      628    dali     737
crush    628    uglab    633
```

Implementing Set Operations

To implement R1 U R2 (while eliminating duplicates) we can

- sort R1 in O(N log N)
- sort R2 in O(M log M)
- merge R1 and R2 in O(N+M)

If we allow duplicates in union (and remove them later) we can

- copy R1 to R3 in O(N)
- insert R2 in R3 in O(M)

If we have an index and don't want duplicates we can

- copy R1 to R3 in O(N)
- for each tuple in R2 (which is O(M))
    - use index to lookup tuples in R1 with the same index value O(1)
    - if R2 tuple equals some such R1 tuple, don't add R2 tuple to R3

Intersection and set difference have corresponding implementations.

Implementing Projection

To implement projection we must

- process every tuple in the relation
- remove any duplicates that result

To avoid duplicates we can

- sort the result and remove consecutive tuples that are equal
    - requires time O(N log N) where N is the size of the original relation
- implement the result as a set

- set insertion guarantees no duplicates
- by using a hash table, insertion is O(1), so projection is O(N)

Implementing Selection

In the absence of an index we

- apply the predicate to every tuple in the relation
- insert matches in the resulting relation
  - duplicates can't occur
- take O(N) time

Given an index, and a predicate that uses the index key, we

- Lookup tuples using the key
- evaluate only those tuples with the predicate
- take O(K) time, where K tuples match the key

Implementing Join with Nested Loops

A *nested loop join* on relations R1 (with N domains) and R2 (with M domains), considers all |R1| x |R2| pairs of tuples.


```
R3= join(R1,Ai,R2,Bj)

for each tuple t in R1 do
  for each tuple s in R2 do
    if t.Ai = s.Bj then
      insert(R3, t.A1, t.A2, ..., t.AN,
        s.B1, ..., s.B(j-1), s.B(j+1), ..., s.BM)
```

This implementation takes time O(|R1|*|R2|).

Index Join

An *index join* exploits the existence of an index for one of the domains used in the join to find matching tuples more quickly.

```
R3= join(R1,Ai,R2,Bj)

for each tuple t in R1 do
```

```
for each tuple s in R2 at index(t.Ai) do
   insert(R3, t.A1, t.A2, ..., t.AN,
     s.B1, ..., s.B(j-1), s.B(j+1), ..., s.BM)
```

We could choose to use an index for R2, and reverse the order of the loops.

The decision on which index to use depends on the number of tuples in each relation.

Sort Join

If we don't have an index for a domain in the join, we can still improve on the nested-loop join using *sort join*.

```
R3= join(R1,Ai,R2,Bj)
```


- Merge the tuples of both relations into a single list
  - list elements must identify the original relation
- Sort the list based on the value in the join domains Ai and Bj
  - all tuples on the sorted list with a common value for the join domains are consecutive
- Pair all (consecutive) tuples from the two relations with the same value in the join domains

---

Comparison of Join Implementations

Assumptions

- Join R1 and R2 (on domain D) producing R3
- R1 has i tuples, R2 has j tuples
- |R3| = m, 0 <= m <= i * j
- Every implementation takes at least time O(m)

Comparison

- Nested-loop join takes time O(i * j)
- Index join (using R2 index) takes time O(i+m)
  - lookup is O(1) for each tuple in R1
  - at most O(m) tuples match
- Sort join takes time O(m +(i+j)log(i+j))
  - O(i+j) to merge the tuples in R1 and R2
  - O((i+j) log (i+j)) to sort the list
  - O(m) to produce the output (0 <= m <= i*j)

Expressing Queries in Relational Algebra

Relational algebra is an unambiguous notation (or formalism) for expressing queries.

Queries are simply expressions in relational algebra.

Expressions can be manipulated symbolically to produce simpler expressions according to the laws of relational algebra.

Expression simplification is an important query optimization technique, which can affect the running time of queries by an order of magnitude or more.

- early "selection" reduces the number of tuples
- early "projection" reduces the number of domains

Algebraic Laws for Join

Commutativity (assuming order of columns doesn't matter)

join(R1, Ai, R2, Bj) = join(R2, Bj, R1, Ai)

Nonassociativity

join (join(R1, Ai, R2, Bj),Bj,R3,Ck)

is not the same as

join (R1,Ai,join(R2, Bj, R3, Ck),Bj)

Algebraic Laws for Selection

Commutativity

select(select(R1,P1),P2) = select(select(R1,P2),P1)

Selection pushing

- if P contains attributes of R

  select(join(R,Ai,S,Bj),P) = join(select(R,P),Ai,S,Bj)

- if P contains attributes of S

select(join(R,Ai,S,Bj),P) = join(R,Ai,select(S,P),Bj)

Selection Splitting (where P = A and B)

select(R,P) = select(select(R,A),B)

select(R,P) = select(select(R,B),A)

Example: Selection Pushing and Splitting

Consider the following 4 relation database

- CSG: Course-StudentID-Grade
- SNAP: StudentID-Name-Address-Phone
- CDH: Course-Day-Hour
- CR: Course-Room

Implement the query "Where is Amy at Noon on Monday?"

Let P be (Name="Amy" and Day="Monday" and Hour="Noon")

We can use a brute-force approach that *joins* all the data in the relations into a single large relation, *selects* those tuples that meet the query criteria, and then isolates the answer field using *projection*.

- R1 = join(CSG,SNAP)
- R2 = join(R1,CDH)
- R3 = join(R2,CR)
- R4 = select(R3,P)
- R5 = project(R4,Room)

```
project(select(join(join(join(CSG,SNAP),CDH),CR),P),Room)
```

The selection uses only Name, Day, and Hour attributes (and not Course or Room), so we can push the selection inside the outermost join.

- R1 = join(CSG,SNAP)
- R2 = join(R1,CDH)
- R3 = select(R2,P)
- R4 = join(R3,CR)
- R5 = project(R4,Room)

We cannot push selection further, because the predicate involves attributes from both operands of the next innermost join (R1,CDH).

We can split the selection into two, one based on Name, and the other based on Day-Hour.

- R1 = join(CSG,SNAP)
- R2 = join(R1,CDH)
- R3 = select(R2,Day="Monday" and Hour="Noon")
- R4 = select(R3,Name="Amy")
- R5 = join(R4,CR)
- R6 = project(R5,Room)

Now we can push the first selection inside the join, since it involves only attributes from the CDH relation.

- R1 = join(CSG,SNAP)
- R2 = select(CDH,Day="Monday" and Hour="Noon")
- R3 = join(R1,R2)
- R4 = select(R3,Name="Amy")
- R5 = join(R4,CR)
- R6 = project(R5,Room)

Similarly we can push the second selection inside the preceding join, since it involves only attributes from R1 (ie, Name).

- R1 = join(CSG,SNAP)
- R2 = select(CDH,Day="Monday" and Hour="Noon")
- R3 = select(R1,Name="Amy")
- R4 = join(R2,R3)
- R5 = join(R4,CR)
- R6 = project(R5,Room)

Continuing to push the second select inside the first join

- R1 = select(SNAP,Name="Amy")
- R2 = join(CSG,R1)
- R3 = select(CDH,Day="Monday" and Hour="Noon")
- R4 = join(R2,R3)
- R5 = join(R4,CR)
- R6 = project(R5,Room)

Algebraic Laws for Projection

Projection pushing

To push a projection operation inside a join requires that the result of the projection contain the attributes used in the join.

project(join(R,Ai,S,Bj),D1,D2,...Dn)

In this case, we know that the domains in the projection will exist in the relation that results from the join.

In performing projection first (on the two join relations)

- we should only project on those domains that exist in each of the two relations
- we must ensure that the join domains Ai and Bj exist in the resulting two relations

Let PDR = {D|D domain in R, D in {D1...Dn}} U Ai

Let PDS = {D|D domain in S, D in {D1...Dn}} U Bi

R1 = project(R,PDR)

R2 = project(S,PDS)

R3 = join(R1,Ai,R2,Bj) = project(join(R,Ai,S,Bj),D1,D2,...Dn)

Example: Projection Pushing

Implement the query "Where is Amy at Noon on Monday?"

- R1 = select(SNAP,Name="Amy")
- R2 = join(CSG,R1)
- R3 = select(CDH,Day="Monday" and Hour="Noon")
- R4 = join(R2,R3)
- R5 = join(R4,CR)
- R6 = project(R5,Room)

This approach carries along unnecessary attributes every step of the way.

- R1 carries Address and Phone attributes
- R4 carries Grade attribute

We use projection pushing to eliminate unnecessary attributes early in the implementation.

- R1 = select(SNAP,Name="Amy")
- R2 = join(CSG,R1)
- R3 = select(CDH,Day="Monday" and Hour="Noon")
- R4 = join(R2,R3)
- R5 = project(CR, Course, Room)
- R6 = project(R4, Course)
- R7 = join(R5,R6)
- R8 = project(R7,Room)

Note that R5 is unnecessary, since the domains in the projection are all the domains of CR.

Implement the query "Where is Amy at Noon on Monday?"

- R1 = select(SNAP,Name="Amy")
- R2 = join(CSG,R1)
- R3 = select(CDH,Day="Monday" and Hour="Noon")
- R4 = join(R2,R3)
- R5 = project(R4, Course)
- R6 = join(CR,R5)
- R7 = project(R6,Room)

We can continue pushing the projection on Course below the join for R4.

- R1 = select(SNAP,Name="Amy")
- R2 = join(CSG,R1)
- R3 = select(CDH,Day="Monday" and Hour="Noon")
- R4 = project(R2,Course)
- R5 = project(R3,Course)
- R6 = join(R4,R5)
- R7 = join(CR,R6)
- R8 = project(R7,Room)

We can continue pushing the projection on Course for R4 below the join for R2.

- R1 = select(SNAP,Name="Amy")
- R2 = project(CSG,Course,StudentID)
- R3 = project(R1,StudentID)
- R4 = join(R2,R3)
- R5 = project(R4,Course)
- R6 = select(CDH,Day="Monday" and Hour="Noon")
- R7 = project(R6,Course)
- R8 = join(R6,R7)
- R9 = join(CR,R8)
- R10 = project(R9,Room)

Relational calculus

An operational methodology, founded on predicate calculus, dealing with descriptive expressions that are equivalent to the operations of relational algebra. Codd's reduction algorithm can convert from relational calculus to relational algebra.
Two forms of the relational calculus exist: the tuple calculus and the domain calculus.
predicate calculus is the system of symbolic logic concerned not only with relations between propositions as wholes but also with the representation by symbols of individuals and predicates in propositions and with quantification over individuals Also called functional calculus

In contrast with Relational Algebra, Relational Calculus is non-procedural query language, that is, it tells what to do but never explains the way, how to do it.
Relational calculus exists in two forms:

The Tuple Relational Calculus [TRC]

1. The tuple relational calculus is a nonprocedural language. (The relational algebra was procedural.)

   We must provide a formal description of the information desired.

2. A query in the tuple relational calculus is expressed as

   $$\{t \mid P(t)\}$$

   i.e. the set of tuples t for which predicate P is true.

3. We also use the notation
   - $t[a]$ to indicate the value of tuple t on attribute a.
   - $t \in r$ to show that tuple t is in relation r.

Tuple relational calculus Implementation

Filtering variable ranges over tuples
Notation: { T | Condition }
Returns all tuples T that satisfies condition.

For Example:

```
{ T.name | Author(T) AND T.article = 'database' }
```

Output: returns tuples with 'name' from Author who has written article on 'database'.
TRC can be quantified also. We can use Existential (∃)and Universal Quantifiers (∀).

For example:

```
{ R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}
```

Output : the query will yield the same result as the previous one.

The Domain Relational Calculus [DRC]

Domain variables take on values from an attribute's domain, rather than values for an entire tuple.

A formal query in DRC is expressed as {<x1, x2 , …, xn> | P(x1, x2 , …, xn)} where x1, x2 , …, xn are domain variables and P(x1, x2 , …, xn) is a formula involving those variables.

Formal definition of DRC formula

Similar to the TRC, formula in Domain Relational Calculus is built up from atoms. An atom in the Domain Relational Calculus has one of the following forms:

- $< x1 , x2 , . . . , xn > \in r$ where r is a relation of n attributes and x1, x2 , …, xn are domain variables or domain constant.

- x op y where x, y are domain variables , op is a comparison operation. Note that x, y have domains that can be compared by op
- x op const where x is a domain variable and const is a constant in domain of attribute for which x is a variable.

The following rules are used to build up the Domain Relational Calculus formula from atoms:

- An atom is a formula
- If P is a formula , then so are ¬P and (P)
- If P1 and P2 are formulae, then so are P1∧P2 , P1∨P2and P1 ⇒ P2
- If P(t) is a formula in x where x is a domain variable then ∃x (P(x)) and ∀x(P(x)) are also formulae.

Domain relational calculus implementation
In DRC the filtering variable uses domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation:

{ a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

where a1, a2 are attributes and P stands for formulae built by inner attributes.

For example: {< article, page, subject > | ∈ TutorialsPoint ∧ subject = 'database'}

Output: Yields Article, Page and Subject from relation TutorialsPoint where Subject is database.
Just like TRC, DRC also can be written using existential and universal quantifiers. DRC also involves relational operators.
Expression power of Tuple relation calculus and Domain relation calculus is equivalent to Relational Algebra.

Example queries

Query 1: Find all employees whose salary is greater than 30.000