Relational Algebra Queries

R1 Reserves

Sid	bid	day
22	101	10/10/18
58	103	10/5/17

Boats

bid	Color
101	Green
103	Red
102	blue
104	Green

S1 (Sailors1)

Sid	Sname	rating	age
22	Dustin	7	45
31	Lubber	8	35
58	Rusty	10	55

S2 (Sailors2)

Sid	Sname	rating	age
28	Yamuna	9	38
31	Lubber	8	35
44	Kriti	6	28
58	Rusty	10	55

(1) Find name of sailors who have reserved boat 103.

Sol1:- Π_{Sname} ($\sigma_{bid=103}$ (Reserves) $\bowtie$ Sailors))
S13

Ans

Sname
Rusty

Sol2:- P (Temp1, $\sigma_{bid=103}$ Reserves)

P (Temp2, Temp1 $\bowtie$ Sailors1

Π_{Sname} (Temp2)

Count (Stuid).

(ii) Find name of sailor who reserves a red boat.

Sol1:-

$$\Pi_{Sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors)$$

Sol2:-

~~P(Temp1, σ~~

$$\Pi_{Sname}(\Pi_{Sid}((\Pi_{bid}\ \sigma_{color='red'} Boats) \bowtie Reserves) \bowtie Sailors)$$

(iii) Find sailor who reserved red or green boat

$$P(Temp, \Pi_{Sname}(\sigma_{color='red' \vee color='green'} Boats))$$

$$\Pi_{Sname}(Temp, \bowtie Reserves \bowtie Sailors)$$

iv) Find Sailor who reserved red and green boat

$$P(Temp_{red}, \Pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves)$$

$$P(Temp_{green}, \Pi_{sid}((\sigma_{color='green'} Boats) \bowtie$$

$\Pi_{sname}((Temp\,green \cap Temp\,red) \bowtie Sailors)$

Student (SSn, Name, Subject, DOB)
Course (C_id, Name, Dept)
Enroll (ssn, c_id, Semester, Grade)
Book_issued (C_id, Semester, ISBN)
Text (ISBN, Title, publisher, Author)

(i) Select all course available in institute

$\Pi_{cid, cname, Dept}$ (Course)

ii) Find all students details registered for course id 10

$\Pi_{ssn, name}(\sigma_{cid=10}(Enroll \bowtie Student))$

iii) Find various book title & author for semester higher than 3.

$\Pi_{ISBN, Title, author}(\sigma_{semester>3}(Book\_issued \bowtie Text))$

iv) Find all student belong to IT (without join)

→ Find c-id of dep = 'IT'

$P(T_1, \Pi_{cid}(\sigma_{Dept='IT'}(Course))$

→ Student Enrolled for the above c_id

$P(T_2, \Pi_{ssn}(\sigma_{cid} Enroll \bowtie T_1)$

→ $\Pi_{ssn, Name, DOB}(Student \bowtie T_2)$

Another way

E_no	name	Salary	dept
1		10000	IT
2		5000	HR
3		40000	IT
4		3000	Marketing
5		3000	

To find Sum of salary of Employee

$$F_{sum(Salary)}(Employee)$$

To find Count of salary
ie to return no of records

$$F_{count(Salary)}(Employee)$$

To find distinct salary

$$F_{count(Salary)}(\Pi_{salary}(Employee))$$

Find average salary in each dept

$$dept\ F_{avg(Salary)}(Employee)$$

Depositor (cname, ano)
Acc (A_no, bname, bal)
Branch (bname, bcity, assets)
Customer (Cname, CStreet, City)

Find sum of asset all branches

$G_{sum(assets)}$ (branch)

Find the count distinct Customer name having accounts

$G_{count\ distinct\ (cname)}$ (Depositor)

Find branch city wise max assets

bcity $G_{max\ (assets)}$ (branch)

bname	bcity	assets
Hetul	Navimu	1000
Vashi		2000
Sanpada		3000
Panvel	Raigad	5000
Pen	"	6000

bcity	max assets
Navi Mumbai	3000
Raigad	6000

Employee-name	branch-name	Salary
Thomsen	Neeul	1500
Lorea	Neeul	2050
Lilly	Vashi	3000
Rao	Neeul	1580
Gopi	Vashi	4500
Adam	Sanpada	9500
Girish	Sanpada	8600

(I) Sum salary (Employee)

(II) Count - distinct branch-name (employee)

(III) branch-name G sum salary (Employee)

Thomsen	Neeul	1500
Lorea	Neeul	2050
Rao	Neeul	1580
Lilly	Vashi	3000
Gopi	Vashi	4500
Adam	Sanpada	9500
Girish	Sanpada	8600

Neeul	6030
Vashi	7500
Sanpada	18000

(IV) branch name G sum salary max Sal (Employee)

Neeul	6030	2050
Vashi	7500	4500
Sanpada	18000	9500

Relational Calculus

* Declarative language

* based on symbolic language ie predicate Calculus

* two types of query languages
  (1) TRC - TUple Relational Calculus
  DRC - Domain " "

* Relational Calculus
  - User define queries in terms of what result they want not how to compute

* It has Variables, Constants, Comparisor operators, logical Conectives & quantifiers

Predicate Logic :- An expression of one or more variables defined on some specific domain. A predicate can be made a proposition by either assigning a value to variable or by quantifying the variable

Let $E(X,Y)$ denote $x = y$
$X(a,b,c)$ denote $a+b+c=0$

Universal quantifiers uses implications
Existential " uses Conjunction

TRC

* Non procedural query language in form

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

tuple

t - Variables

P(t) - predicate (P) true for t

$t \in r$ - t belongs to relation

t[A] - Value in tuple t

P(t) is formula Contains

* attributes & Constants
* Comparison operators ($<, >, \geq, \leq, =, \neq$)
* Connectives, $\neg, \wedge, \vee$
* Implications $x \Rightarrow y$; x true then Y is true.

Quantifiers

$\exists$, there exist

$\forall$, P(t) true for all $t \in r$

(i) To find details of employee who earns more than 25000

$$\{e \mid Employee(e) \wedge e.Salary > 25000\}$$

(ii) To find Salary of employee

$$\{e.Salary \mid Employee(e) \wedge e.Salary > 25000\}$$

SQL to TRC translation

Select attributes to go on left side of | character
From relation mapped to tuple variable & combined using AND
Where append to tuple variable with AND

Universal quantifier

$\forall x\, p(x)$ — for every value of x, p(x) is true

Existential quantifier

$\exists x (p(x))$ — some value of x, p(x) is true

General Condition for tuple

$\{t_1.A_j, t_2.A_k \ldots \mid Cond(t_1, t_2 \ldots)\}$

(1) Find rollno, name of all students in deptno = 2

Student (rollno, name, dept_no, gender)

$\{S.rollno, S.name \mid Student(t) \wedge t.dept\_no = 2\}$

To write Complex queries quantifier is used.

Quantifier — logical symbol which makes an assertion about set of values which make one or more formulas true

(1) Using quantifier to TRC query

emp (Eid, name, address)
Dependent (Did, name, eid)

Employee name who do not have dependent

$$\{e.name \mid emp(e) \wedge \neg(\exists d\ (dept(d) \wedge (d.Eid = e.Eid))\}$$

tuple var with free of quantifier is free variable

to be retrieved by query

tuple var with quantifier / to make assertion is called bound variable

$$(\forall x)p(x) = \neg(\exists x)(\neg p(x))$$

Using universal quantifier

$$\{e.name \mid employee(e) \wedge (\forall d\ \neg(Dependent(d) \wedge (e.eid = d.eid)))\}$$

↓

$$\{e.name \mid employee(e) \wedge (\forall d(\neg Dept(d) \vee \neg(e.eid = d.eid)))\}$$

(i) Find loan no, branch, amt of loan greater than or equal to 10,000

TRC query

$\{t \mid t \in loan \wedge t.amount >= 10000\}$

O/P

Lno	Bname	Amt
L33	ABC	10000
L35	DEF	15000
L98	DEF	65000

(ii) Find name of customer who have loan and account in bank

$\{t \mid \exists b \in borrower (t.Customername = b.Customername) \wedge \exists d \in deposit (t.Customername = d.Customername)\}$

O/P

[(Saurabh, Mehak, Ria) (Saurab, Mehak Ria)]

$\wedge$ (Saurab, Mehak, Sumiti)
(Saurabh, Mehak, Sumiti)

Finally

Saurabh
Mehak.

Transforming Universal & Existential quantifiers

$\forall(x)(p(x) \equiv \neg(\exists x(\neg(p(x))))$

$\exists(x)(p(x) \equiv \neg(\forall x(\neg(p(x))))$

$(\forall x)(p(x)$ AND $Q(x)) \equiv$ NOT $(\exists x)$ (NOT $(p(x))$ OR NOT $(Q(x$

$(\forall x)(p(x)$ OR $Q(x)) \equiv$ NOT $(\exists x)$ (NOT $(p(x))$ AND NOT $(Q(x))$)

$(\exists x)(p(x))$ OR $Q(x)) \equiv$ NOT $(\forall x)$ (NOT $(p(x))$ AND NOT $(Q(x))$)

$(\exists x)(p(x)$ AND $Q(x)) \equiv$ NOT $(\forall x)$ [NOT $(p(x))$ OR NOT $(Q(x))$]

$(\forall x)(p(x)) \Rightarrow (\exists x)(p(x))$

NOT $(\exists x)(p(x)) \Rightarrow$ NOT $(\forall x)(p(x))$

Relational algebra Queries & TRC queries are same powerful.

(i) Find name & ages of sailors with rating above 7.

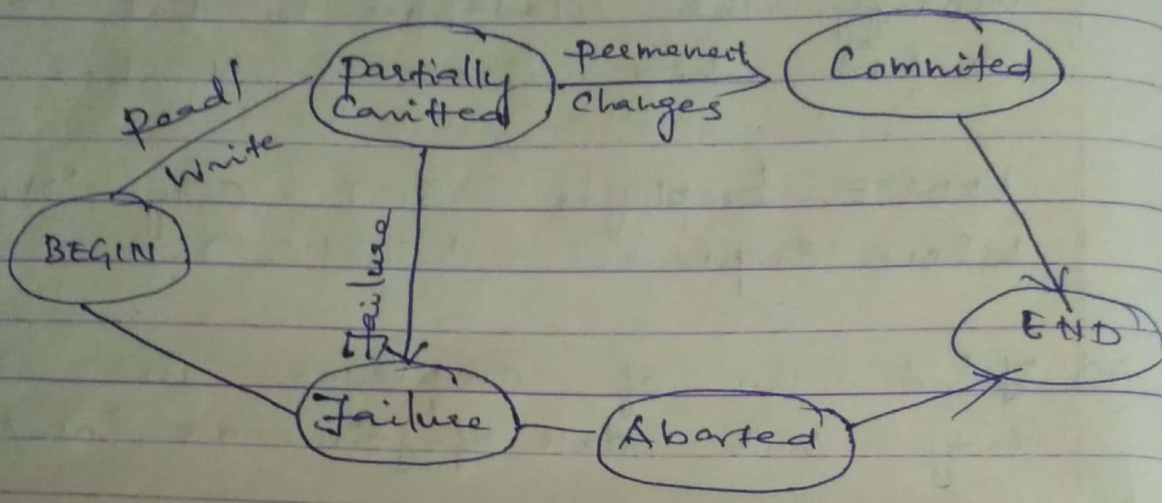
$\{S.age, S.name \mid S \in Sailors \wedge S.rating > 7\}$

ii. Find S name ~~and~~ bid and reservation date for each reservation

$\{t \mid \exists R \in Reserves \ \exists S \in Sailors$

$(R.Sid = S.Sid \wedge t.bid = R.bid \wedge t.day = R.day \wedge t.Sname = S.Sname)\}$

Transaction state :-



Transaction failures

Logical Errors : Some Error Codes / internal Error

System Errors : Deadlock, resource unavailability

System Crash - Interruption in power failure

Disk failure - Unreadable of disk, head Crash.

Transaction

Definition
Transaction is a logical unit of program which involves sequence of steps executes once and preserves Consistency.

Properties
* Atomicity — All are none
* Consistency — Transaction Complete or aborted Consistency must ensured
* Isolation — No other transaction must view partial result
* Durability — Transaction Comited will be saved permanent in DB.

Transaction Ex:-

BEGIN
Step1:- Read acc A from disk
Step2:- If bal < 100, return proper msg
Step3:- Subtract bal − 100
Step4:- Write A to disk
Step5:- Read acc B from disk
Step6:- Add bal +100
Step7:- Write acc B to disk

TRANSFER FROM ACC-A to ACC-B.

Sys fails in step 5, 6, inconsistent of data happens.
Roll back is needed.

Read set:- Set of items read by transaction Called read set
Write set:- Set of items write by transaction Called Write set.

Conflict Situations:-

(i) Write set of one transaction T_1 & intersect with read set of another transaction T_2. Whether W or R done first. Conflict occurs Called RW Conflict

(ii) W of T_1 intersect W of T_2. Conflict occurs. Called WW Conflict.

Concurrency Anamolies:-

(1) Lost Updates

Transaction A	Time	Transaction B
Read N	1	
	2	Read N
N: N-5	3	
	4	N: N-4
Write N	5	
	6	Write N.

Airline Reservation to reserve seats on flights.

Inconsistent Retrieval:-

T1 & T_2 accessing department DB

T_1 — Updates DB to give all employees 3% raise in salary

T_2 — Total Salary bill of department.

T_1	Time	T_2
Read Employee 100	1	
	2	Sum = 0.0
Update Salary	3	
	4	Read employee 100
Write employee 100	5	
	6	Sum = Sum + Salary
Read Employee 101	7	
Update Salary	8	
Write Employee 101	9	
	10	Read Emp 101
	11	Sum = Sum + Sal

Uncommited Dependency:-

Transaction1	Time	T2
	1	Read Q
	2	Update Q
	3	Write Q
Read Q	4	
	5	Read R
Update Q	6	
Write Q	7	
	8	Failure (roll back)

Schedule:-

Process Creating a single group of multiple parallel transactions & executing one by one.

Serial Schedule:-

Operation of one Transaction are Completed before another T begins

Schedule

Serial | Non Serial | Serializable

(a)

T_1	T_2
Read (A)	
A: (A) - N	
Write (A)	
Read (B)	
B: B+N	
Write (B);	Read (A);
	A = A+m;
	Write (A)

schedule (A)

(b)

T_1	T_2
	read (A)
	A = A+m
	Write (A)
Read (A)	
A: A - N;	
Write (A);	
Read (B);	
B: B+N;	
Write (B);	schedule (B)

Non Serial

T_1	T_2	A = 100
read (A)		
A: A - N;		
(50)		
	Read (A)	
	A: A + M;	100+50
Write (A);		
read (B);		
	Write (A);	(c)
B: B+N		
Write (B);		

T_1	T_2
read (A)	
A: A-N;	
Write (A);	
	Read (A)
	A: A+m;
	Write (A);
Read (B)	
B: B+N;	
Write (B);	

A non serial schedule will be Bf serializable, if result is equal to result of serial executed transactions.

Non - serial
- Strict
- Cascadeless
- Recoverable

Strict

T_a	T_b
R(x)	
	R(x)
W(x)	
Commit	
	W(x)
	R(x)
	Commit

Write operation of transaction T_a precedes the read/Write operation of T_b so that Commit or abort of transaction T_a should also precede read/ Write of T_b.

Cascadeless

T_a	T_b
R(x)	
W(x)	
	W(x)
Commit	
	R(x)
	W(x)
	Commit

If transaction is going to perform read operation on value, it has to wait until the transaction who perform Write on value to Commit.

Recoverable

T_a	T_b
R(x)	
W(x)	
	R(x)
	W(x)
	R(x)
Commit	
	Commit

Current transaction Commits only after the Commit of transaction which is updating value.

Serializable Schedule:-

Serializable schedules always leaves DB in Consistent state.
Serial schedule ⟶ Serializable

Types:-
Conflict — Checks whether non serial schedules are
View Conflict/serializable or not

Conflict serializability:-

If we can convert non-serial into serial schedule after swapping its non-conflicting operations

Conflicting operations

(i) Both operation should belong to diff transaction
(ii) both should work on same data
(iii) Atleast one operation must be write.

Ex:-

T_1	T_2
R(A)	
R(B)	
	R(A)
	R(B)
	W(B)
W(A)	

To convert non-serial to serial need to swap T_2 R(A) with W(A) of T_1

Cannot Swap R(A) of T_2 with W(A) of T_1 ∴ it is Conflicting operations. , So it is not Conflict Serializable.

Ex:-2

T_1	T_2
R(A)	
	R(A)
	R(B)
	W(B)
R(B)	
W(A)	

Step 1 :-

Swap R(A) → T_1 to R(A) T_2

T_1	T_2
	R(A)
R(A)	
	R(B)
	W(B)
R(B)	
W(A)	

Step 2 :- Swap R(A) → T_1 to R(B) → T_2

T_1	T_2
	R(A)
	R(B)
R(A)	
	W(B)
R(B)	
W(B)	

Step3:- Swap W(B) T_2 to R(A) T_1

T_1	T_2
	R(A)
	R(B)
	W(B)
R(A)	
R(B)	
W(B)	

Step:-4 Now Converted into Serial schedule so it is Conflict Serializable.

Ex:- To solve in class

S_1 : $R_1(x)$ $R_1(y)$ $R_2(x)$ $R_2(y)$ $W_2(y)$ $W_1(x)$

S_2 : $R_1(x)$ $R_2(x)$ $R_2(y)$ $W_2(y)$ $R_1(y)$ $W_1(x)$

T_1 = $R_1(x)$ $R_1(y)$ $W_1(x)$
T_2 = $R_2(x)$ $R_2(y)$ $W_2(y)$

Check Serializability for S_1

$R_1(x)$
$R_1(y)$
$R_2(x)$
$R_2(y)$
$W_2(y)$
$W_1(x)$

T_1 followed by T_2
$\therefore$ $R_2(x)$ must be swapped by $W_1(x)$ but they are Conflicting. So S_1 is not Conflict Serializable.
(Can't Convert to Serial)

S_2 : $R_1(x)$ $R_2(x)$ $R_2(y)$ $W_2(y)$ $R_1(y)$ $W_1(x)$

T_1 $R_1(x)$ $R_1(y)$ $W_1(x)$
T_2 : $R_2(x)$ $R_2(y)$ $W_2(y)$

	T_1	T_2
$S_2 \rightarrow$	$R_1(x)$	
	~~$R_2(x)$~~	$R_2(x)$
	~~$R_2(y)$~~	$R_2(y)$
		$W_2(y)$
	$R_1(y)$	
	$W_1(x)$	

Step1:- Swapping $R_1(x)$ with $R_2(x)$

	T_1	T_2
$S_2' \rightarrow$		$R_2(x)$
	$R_1(x)$	
		$R_2(y)$
		$W_2(y)$
	$R_1(y)$	
	$W_1(x)$	

Step2' $S_2'' \rightarrow$

T_1	T_2
	$R_2(x)$
~~$R_1(y)$~~	$R_2(y)$
$R_1(x)$	
	~~$W_2(y)$~~
$R_1(y)$	
$W_1(x)$	

$S_2''' \rightarrow$

T_1	T_2
	$R_2(x)$
	$R_2(y)$
	$W_2(y)$
$R_1(x)$	
$R_1(y)$	
$W_1(x)$	

$T_2 \rightarrow T_1$ S_2 is Conflict

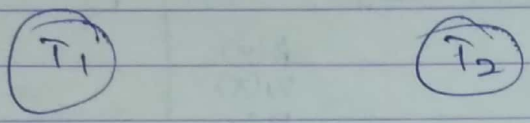
Precedence graph for testing Conflict Serializability

(i) Create node T in graph
(ii) For Conflicting operation $r(x)$ & $w(x)$, $w(x)$ & $r(x)$, $w(x)$, $w(x)$ draw an edge from T_i to T_j

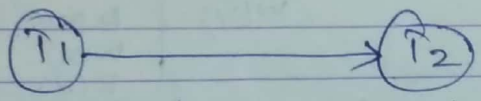
Schedule Ex:-

$S: r_1(x)\ r_1(y)\ W_2(x)\ W_1(x)\ r_2(y)$

Step 1.

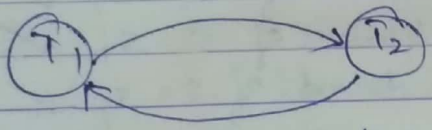


Step 2:-



$r_1(x)$ happens before $W_2(x)$

Step 3:-



$W_2(x)$ happens before $W_1(x)$

∴ Cycle exist, so Schedule is non-Serializable.

T_1	T_2	T_3	T_4
R(A)	R		
	R(A)		
		R(A)	
			R(A)
W(B)			
	W(B)	W(B)	
			W(B)

T_1	T_2	T_3	T_4
R(A)			
W(B)			
	R(A)	R(A)	R(A)
	W(B)	W(B)	W(B)

View Equivalent :

T_1	T_2
R(x)	
W(x)	
	R(x)
	W(x)
R(Y)	
W(Y)	
	R(Y)
	W(Y)

Serial →

T_1	T_2
R(x)	
W(x)	
R(Y)	
W(Y)	
	R(x)
	W(x)
	R(Y)
	W(Y)

Initial read of $S = S_1$ ✓
Final Write of $S = S_1$ ✓
Update Read of $S = S_1$ ✓

S

∴ R(x) initial in T_1 = R(x)
W(Y) final Write $T_2 = S_1$
Update T_2 reads value of x written by T_1 = Same

Every Conflict Serializable is View Seridlizable.

Every View Serializable which is not Conflict has blind writes.