CS3DB3/SE4DB3/SE6DB3 TUTORIAL

YU HUANG

2015-03-25

Outline

- assignment 2 explanation
- relational algebra
- serial schedule, conflict serializability, ACA, strict schedule, recoverable
- Locks, strict 2PL, 2PL

Relational Algebra

- select $R1 := \sigma_C(R2)$
- projection $R1 := \pi_L(R2)$
- theta-join $R3 := R1 \bowtie_C R2$ take the product R1 X R2, then apply $\sigma_C(R1 \times R2)$
- rename ρ
- sort tuples τ
- eliminate duplicates δ ; grouping γ

3.Determine the number of unique products that have an extended warranty.

```
--3: Determine the number of unique products that have an extended warranty.
select count (p.ID) as numExtendedWarrantyProds
from product p, warranty w, with
where with.productID = p.ID
        and with.warrantyID = w.ID
        and w.type = 'Extended';
```

3. $\gamma_{COUNT(product.ID) \rightarrow numExtendedWarrantyProds}(\sigma_{type='Extended'}(Product \bowtie_{Product.ID=With.productID} With \bowtie_{With.warrantyID=Warranty.ID} Warranty))$

7a.List the vendor (ID, name) that sold greater than 10 (quantity) products each day. Order your results in ascending order by date and the total quantity sold. Show the total number of products sold for each vendor.

```
select v.ID, v.name, SUM(t.quantity) as numProducts, t.Date
from Vendor v, Transaction t
where v.ID = t.vendorID
group by v.ID, v.name, t.Date
having SUM(t.quantity) > 10
order by t.Date, sum(t.quantity);
```

7a. $\tau_{date,numProducts}(\pi_{Vendor.ID,name,numProducts,date}(\sigma_{numProducts>10} (\gamma_{Vendor.ID,name,SUM(quantity) \rightarrow numProducts,date}(Vendor \bowtie_{Vendor.ID=Transaction.vendorID} Transaction))))$

11.Find all attendees (ID, first name, last name) that purchased at least one product each day of the three day event (April 2 - 4, 2014).

```sql
SELECT p.ID, p.firstName, p.lastName
FROM person p
WHERE p.ID in (select t1.attendeeID
        from Transaction t1, Transaction t2, Transaction t3
        where t1.attendeeID = t2.attendeeID
        and t2. attendeeID = t3.attendeeID
        and t1.attendeeID = t3. attendeeID
        and t1.Date = '04/02/2014'
        and t2.Date = '04/03/2014'
        and t3.Date = '04/04/2014');
```

11. $\pi_{Person.ID, firstName, lastName}(Person \bowtie_{Person.ID=bt.attendeeID} \rho_{bt}(\pi_{t1.attendeeID}$
$(\sigma_{t1.date='04/02/2014' \ \wedge \ t2.date='04/03/2014' \ \wedge \ t3.date='04/04/2014'}(\rho_{t1}(Transaction)$
$\bowtie_{t1.attendeeID=t2.attendeeID} \rho_{t2}(Transaction) \bowtie_{t2.attendeeID=t3.attendeeID} \rho_{t3}(Transaction)))))$

12.Determine the highest total sales for a product sub-category over the entire three day event.

```sql
SELECT MAX(totalSales)
FROM
(
SELECT ps.name as groupName, SUM(t.amount) as totalSales
FROM ProductSubCategory ps, Product p, Transaction t, ProductBelongToSubcategory pbs
WHERE pbs.productSubCategoryID = ps.ID
    and pbs.productID = p.ID
    and p.ID = t.productID
GROUP BY ps.name);
```

12. $\gamma_{MAX(totalSales)}(\gamma_{ProductSubCategory.name, SUM(amount) \rightarrow totalSales}$
$(ProductSubCategory \bowtie_{ProductSubCategory.ID=ProductBelongToSubcategory.productSubCategoryID}$
$ProductBelongToSubcategory \bowtie_{ProductBelongToSubcategory.productID=Product.ID}$
$Product \bowtie_{Product.ID=Transaction.productID} Transaction))$

Transaction

- Serial Schedule

S: R1(A),W1(A),R1(B), W1(B), R2(A), W2(A), R2(B), W2(B)

T1: R1(A),W1(A),R1(B), W1(B)

T2: R2(A), W2(A), R2(B), W2(B)

- Serializable Schedule, a schedule that is equivalent to a serial schedule.

T1	T2
R(A)	
A := A+100	
	R(A)
	A: = A* 2
R(B)	
B := B+100	
	R(B)
	B:= B*2

- Conflict Serializable

Schedule S is conflict serializable if S is conflict equivalent to some serial schedule

If two schedules S1 and S2 are conflict equivalent then they have the same effect

S1 ←→ S2 by swapping non-conflicting ops

- **R1(A); W1(A); R2(A); W2(A); R1(B); W1(B); R2(B); W2(B)**

Can we transform into a serial schedule by swapping of adjacent non-conflicting actions?

R1(A); W1(A); R1(B); W1(B); R2(A); W2(A); R2(B); W2(B)

T1,T2 are conflicting operations, since the result depends on the action order. Schedule G is conflict equivalent to Schedule <T1,T2>.

$$G = \begin{bmatrix} T1 & T2 \\ R(A) & \\ & R(A) \\ W(B) & \\ Com. & \\ & W(A) \\ & Com. \end{bmatrix}$$

is conflict-equivalent to the serial schedule <T1,T2>, but not <T2,T1>. It means G is conflict equivalent to R1(A), W1(B), R2(A), W2(A), but not R2(A), W2(A),R1(A), W1(B).

- Strict Schedule

A schedule S is strict if a value written by T1 is not read or overwritten by other T2 until T1 aborts or commits.

W1(A); W1(B), C1; W2(A); R2(B); C2;

- Avoid cascading abort (ACA)

Aborting a transaction can be done without cascading the abort to other transaction.

Strategy to avoid cascading aborts is to disallow a transaction from reading uncommitted changes from another transaction in the same schedule.

The left one is not ACA the right is ACA.

T1	T2
R (A)	
W (A)	
	R (A)
	W (A)
Abort	

No

T1	T2
R (A)	
W (A)	
Commit	
	R (A)
	W (A)

Yes

- Recoverable

Transaction(T2) commit only after the commit of all other transaction(T1) whose changes it(T2) read.

No

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	Commit
Abort	

Yes

T1	T2
R(A)	
W(A)	
	R (A)
	W (A)
Commit	
	Commit

Locks

S = Shared lock (for read)

X = Exclusive lock (for write)

- only Shared lock is additional. (we can add another shared lock on a transaction that already has one shared lock)

Lock compatibility matrix

	None	**S**	**X**
None	OK	OK	OK
S	OK	OK	Conflict
X	OK	Conflict	Conflict

Strict two Phase Locking

Two rules:

1. Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
2. All locks held by a transaction are released when the transaction completes

Two Phase Locking Protocol (2PL)

Relaxes the 2nd rule of Strict 2PL to allow Xacts to release locks before the end (commit/abort)

A transaction cannot request additional locks once it releases any lock.

Strict 2PL

T1	T2
L(A);	
R(A), W(A)	
	L(A); **DENIED…**
L(B);	
R(B), W(B)	
U(A), U(B)	
Commit;	
	…GRANTED
	R(A), W(A)
	L(B);
	R(B), W(B)
	U(A), U(B)
	Commit;

2PL

T1	T2
X(A), X(B)	
R(A), W(A)	
U(A)	
	X(A)
	R(A), W(A)
	X(B), **DENIED…**
R(B), W(B)	
U(B)	
	..GRANTED
	R(B), W(B)
	U(A), U(B)

difference, Strict 2PL must commit before other transaction locks.