Week 10 Tutorial Notes

▼ Agenda

Q3 → Introduction to Projection and Relational Algebra

Q4 → Division and other operations

Q5 → Assumptions and Min/Max of operations

Q7 → Example relational algebra questions of real schema

Q8 → Interpreting relational algebra

Q10 → Transaction Processing definitions

Q11 → Precedence graph

Q12 → Conflict-serializability

Q13 → View- and conflict-serializability

Relational Algebra

Selection: *Sel[age > 50](table)*

- Selects the rows in some relation that satisfies expression

Projection: *Proj[name, age] (table)*

- Filters relation by its columns a, b, c

Rename: rename[schema](rel)

- Renames the columns of rel with the columns specified in schema

Union: Table1 Union Table2

- The equivalent of a SQL UNION

Intersection: Table1 Intersect Table2

- The equivalent of a SQL INTERSECT

Difference: Table1 - Table2

- Returns tuples that exist in Table1 but don't exist in Table2
- NOT SYMMETRIC / NOT THE COMPLEMENT OF SET INTERSECTION
  - Table1 - Table2 != Table2 - Table1

Product: Table1 x Table2

- Pairwise product between two tables
- For every row in Table1, attach every row in T2

Student

S_id	Name	Class	Age
1	Andrew	5	25
2	Angel	10	30
3	Anamika	8	35

Course

C_id	C_name
11	Foundation C
21	C++

Student X Course

S_id	Name	Class	Age	C_id	C_name
1	Andrew	5	25	11	Foundation C
1	Andrew	5	25	21	C++
2	Angel	10	30	11	Foundation C
2	Angel	10	30	21	C++
3	Anamika	8	35	11	Foundation C
3	Anamika	8	35	21	C++

Division: Table1 Div Table2

- Returns all tuples in T1 which have an exact match for the rows in T2

R=Proj[D,C](r1)

D	C
4	x
4	y
4	z
5	x
5	y
5	z

S=Proj[G](r2)

G
x
y

R / S

name
4
5

Natural join: Table1 Join Table2

- Joins two tables on common attributes that are matched
- E.g T1(ABCD), T2(CDEF) → T1 Join T2 ⇒ AB**CD CD**EF

Theta join: Table1 Join[condition] Table2

- A natural join with a 'where' condition applied

Outer join: Table1 [Full|Left|Right] Outer Join Table2

- Left outer - all tuples from T1 included (T2 non matches set to null)
- Right outer - all tuples from T2 included (T1 non matches set to null)
- Full outer - both (left and right outer join applies)

Transaction Processing

You may be given a schedule like the following:

```
T1:     R(A) W(Z)             C
T2:             R(B) W(Y)       C
```

```
T3: W(A)              W(B)    C
```

Here are some definitions:

- Operation: when you either read or write to an object
  - Eg. R(A) is reading object A
- Transaction: A group of operations
  - Eg. T1 is a transaction
- Schedule: A group of transactions with a certain order to operations

A **serializable schedule** is when you can find a 'sensible' order to transactions executed one after another (Eg. T1,T3,T2).

To identify if a schedule is serializable, you must check conflict serializability and view serializability.

NOTE: if you know a schedule is conflict serializable, then it is also view serializable

Conflict Serializability

Detecting conflict serializability requires us to draw a precedence graph:

1. Scan the schedule left to right and look for conflicting operations between two transactions
   - Conflicting operations are:
     - R(X) vs W(X)
     - W(X) vs R(X)
     - W(X) vs W(X)
2. For each conflicting operation, draw an arrow from one transaction (with the earlier operation) to the other.
3. Once we have finished our precedence graph, we check if there are any loops
   - If there is one or more loops, the schedule is **not** conflict serializable

- If there are no loops, the schedule is conflict serializable

▼ Example

```
T1:      R(A) W(Z)              C
T2:              R(B) W(Y)       C
T3: W(A)                 W(B)     C

Precedence graph: T2→T3→T1
This is conflict-serializable

T1: R(X) R(Y) W(X)          W(X)
T2:              R(Y)          R(Y)
T3:                  W(Y)

Precedence graph: T1→T3↔T2
This is not conflict-serializable
```

View serializability

For each order of transactions, we check if it is ‘view equivalent’ to the original schedule. An order of transactions is view equivalent if all of the following conditions hold:

- For each object, the final Write operation is done by the same transaction as the original schedule
- For each Read operation, it is reading the same state of that object as the original schedule
    - In the original schedule, a Read operation will either be reading the original state of an object or a new state written by a prior Write operation.

If we find just **one** order of transactions that is view equivalent, then the schedule is view-serializable.

▼ Example

T1:R(X) W(X)
T2: R(X) W(X)

T1,T2 This is not view equivalent
T2,T1 This is not view equivalent
This is not view-serializable

T1:R(X) W(X)
T2: W(X)
T3: W(X)

We know that the order must have T3 at the end
as we require the final Write of X to be done
by T3. So, we only check the following orders:

T1,T2,T3 This is view equivalent
T2,T1,T3 This is not view equivalent
This is view-serializable