Ch4. SQL

A. Basic Query

1. SELECT DISTNICT eliminate equivalent rows
2. FROM
3. WHERE AS / = LIKE string matching 'B_%B' _ any one character, % 0 or more

B. Set Operations

4. UNION
5. INTERSECT
6. EXCEPT

C. Comparsion Operators

1. IN
2. EXISTS $\Theta_{eid}(employees) - \Theta_{eid}(dependent)$
3. op ANY $\Theta_{name}(employees \bowtie participate \bowtie project)$
4. op ALL

Ch 5. Schema Refinement & Normal Forms

A. Motivation

Id	name	age	rating	Hourly_wages	Hours_worked
123-22-3666	Peter	48	8	10	40
231-31-5368	Paul	22	8	10	30
131-24-3650	Mary	35	5	7	30
434-26-3751	David	35	5	7	32
612-67-4134	Ada	35	8	10	40

Rating = 8 → Hourly_wages = 10

Rating = 5 → Hourly_wages = 7

☆ Redundant Storage

~ the associations are repeated (storage is not used efficiently)

1. Update Anomalies

~ do not make a similar change to other tuples if a tuple is updated (may cause inconsistency)

2. Insertion Anomalies

~ not allow to insert a tuple unless we know that hourly wage and rating (otherwise, null values may exist)

3. Deletion Anomalies

~ if we delete all tuples with a given rating value (David and Mary), we may lose the associations

• Decomposition

~ replacing the relation schema by two or more relation schemas that each contains a subset of attributes

Id	name	age	rating	Hourly_wages	Hours_worked
123-22-3666	Peter	48	8	10	40
231-31-5368	Paul	22	8	10	30
131-24-3650	Mary	35	5	7	30
434-26-3751	David	35	5	7	32
612-67-4134	Ada	35	8	10	40

subset of attributes

Id	name	age	rating	Hours_worked
123-22-3666	Peter	48	8	40
231-31-5368	Paul	22	8	30
131-24-3650	Mary	35	5	30
434-26-3751	David	35	5	32
612-67-4134	Ada	35	8	40

rating	Hourly_wages
8	10
5	7

Functional dependency:
- rating determines Hourly_wages

B. Functional Dependencies FD

~ a kind of IC that generalizes the concept of a key

Let R be a relation schema and let X and Y be nonempty sets of attributes in R.
We say that an instance *r* of *R* satisfies the FD $X \rightarrow Y$
If the following holds for every pair of tuples *t1* and *t2* in *r*

If t1.X = t2.X, then t1.Y = t2.Y

The notation t1.X refers to the projection of tuple t1 onto the attributes in X

α determines β

	X	Y
	rating	Hourly_wages
t1	8	10
t2	8	10
	5	7
	5	7
	8	10

If t1.X = t2.X = 8

then t1.Y = t2.Y = 10

student_ID	student_name	course_ID	course_name
111	Chan Tai Man	3170	Database
222	Wong Siu Ling	3170	Database
333	Tam Wai Ming	3160	Algorithms
111	Chan Tai Man	3160	Algorithms

- Trivially Preserved

~ $\alpha \rightarrow \beta$ is trivially preserved if any two rows never agree on α value

course_ID → course_name (NOT trivially preserved)

student_ID, course_ID → course_name (trivially preserved)

∵ cannot find two rows with same student_ID and course_ID

- Trivial Dependency

~ $\alpha \rightarrow \beta$ is a trivial dependency if $\beta \subseteq \alpha$

student_name → student_name (trivial dependency)

student_name, course_name → student_name

- Keys

1. α is superkey for R iff $\alpha \rightarrow R$
2. α is candidate key for R iff $\alpha \rightarrow R$ and no γ that is a proper subset of α, $\gamma \rightarrow R$

- Reasoning about FDs

F - a set of functional dependencies

f - an individual functional dependency

f is implied by F if all functional dependencies in F is true, then f is true

e.g. $F = \{ id \to did, did \to office \}$ implies $id \to office$

- Closure of F F^+

~ if the set of all FDs implied by F, then F is a closure of F, denoted as F^+

$F = \{ A \to B, B \to C, C \to D \}$

$F^+ = \{ A \to B, B \to C, C \to D, A \to A, B \to B, C \to C, D \to D, A \to C, A \to D, B \to D \}$

- Armstrong's Axioms

Reflexivity	if $Y \subseteq X$, $X \to Y$ (trivial dependency)
Augmentation	if $X \to Y$, then $XZ \to YZ$
Transitivity	if $X \to Y$ and $Y \to Z$, then $X \to Z$

- Additional Rules

Union	if $X \to Y$ and $X \to Z$, then $X \to YZ$
Decomposition	if $X \to YZ$, then $X \to Y$ and $X \to Z$

- Attribute Closure

~ X^+ - closure of X under F

$F = \{ A \to B, A \to C, CG \to H, CG \to I, B \to H \}$

To compute AG^+ closure = AG $\quad \because A^+ \not\to R$ and $G^+ \not\to R$

closure = ABG $\quad \therefore AG \to R$

closure = ABCG $\quad \therefore$ AG is a candidate key

closure = ABCGH

closure = ABCGHI

C. Normal Form

1NF	1st Normal Form	No repeating data group
2NF	2nd Normal Form	No partial key dependency
3NF	3rd Normal Form	No transitive dependency
BNF	Boyce-Codd Normal Form	Reduce key dependency
4NF	4th Normal Form	No multi-valued dependency
5NF	5th Normal Form	No join dependency

★ 1NF ⊃ 2NF ⊃ 3NF ⊃ BNF ⊃ 4NF ⊃ 5NF

- 1st Normal Form

~ each attribute should be only atomic values

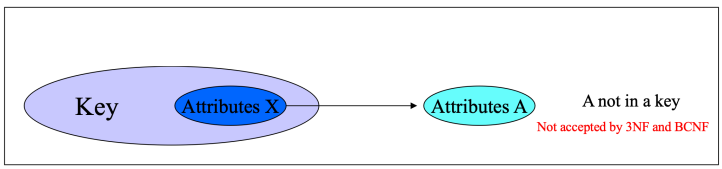
- 2nd Normal Form

~ all non-key attributes are functionally dependent on the primary key

- 3rd Normal Form

~ R is in 3NF if for any $X \to A$ in F,

1. $X \to A$ is a trivial dependency ($A \subseteq X$), OR.
2. X is a superkey for R, OR.
3. A is a part of some key for R

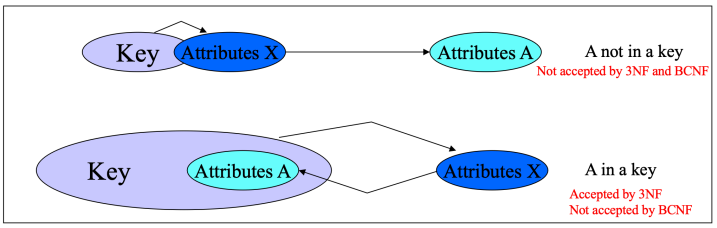


Partial Dependencies

- Boyce-Codd Normal Form

~ R is in BCNF if for any $X \to A$ in F,

1. $X \to A$ is a trivial dependency ($A \subseteq X$), OR.
2. X is a superkey for R



Transitive Dependencies

D. Decomposition

~ A set of relation schemas $\{R_1, R_2, \ldots, R_n\}$ is a decomposition of R if

$$R_1 \cup R_2 \cup \ldots \cup R_n = R$$

Supply(sid, status, city, part_id, qty)

Supplier(sid, status, city)

and

SP(sid, part_id, qty)

⇒ Supply = Supplier ∪ SP

★ Decomposition may turn non-normal form into normal form

Suppose R is not BCNF with FD: $X \to A$ (where $X \cap A = \phi$ ⇒ X cannot be super key) violates the condition

Step 1: Remove A from R,

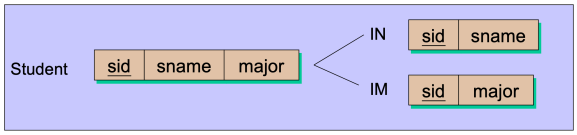
Step 2: Create a new relational schema XA

Repeat until all relations are BCNF

• Lossless Join Decomposition

~ The set of relation schemas $\{R_1, R_2, \ldots, R_n\}$ is a lossless join decomposition of R if

for all possible relation instances r on schema R, $r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \cdots \bowtie \Pi_{R_n}(r)$



Student

sid	sname	major
123	Ling Wang	C.S.
456	Hong Zhou	C.S.

IN

sid	sname
123	Ling Wang
456	Hong Zhou

Student = IN ⋈ IM

IM

sid	major
123	C.S.
456	C.S.

⇒ Student can be recovered by joining the instance of IN and IM

Theorem

the decomposition of R into relations with attribute sets R_1, R_2 is a lossless join decomposition iff

$(R_1 \cap R_2) \to R_1 \in F^+$ ie. $R_1 \cap R_2$ is a superkey for R_1 or R_2

OR. the attributes common to R_1 and R_2 must contain a key for either R_1 or R_2

$(R_1 \cap R_2) \to R_2 \in F^+$

- Dependency Preservation

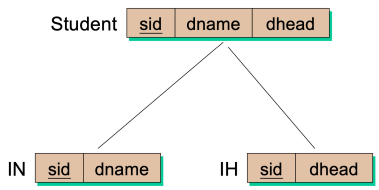
Given relation schema R, Functional Dependencies F and Decomposition $\{R_1, R_2\}$

F_i - the set of dependencies in F^+ involves only attributes in R_i

~ the projection of F on the set of attributes in R_i

★ Dependency is preserved if $(F_1 \cup F_2)^+ = F^+$

Dependency set: F = { sid → dname, dname → dhead }



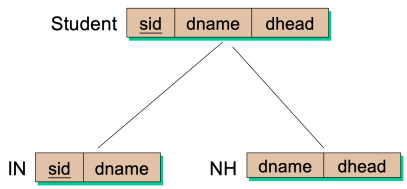
F_{IN} = { trivial dependencies, sid → dname, sid → sid dname }

F_{IH} = { trivial dependencies, sid → dhead , sid → sid dhead }

dname → dhead $\in F^+$ BUT dname → dhead $\notin (F_{IN} \cup F_{IH})^+$

∴ does not preserve dependency

Dependency set: F = { sid → dname, dname → dhead }



F_{IN} = { trivial dependencies, sid → dname, sid → sid dname }

F_{NH} = { trivial dependencies, dname → dhead , dname → dname dhead }

dname → dhead $\in F^+$ AND dname → dhead $\in (F_{IN} \cup F_{NH})^+$

∴ preserved dependency

- BCNF Decomposition

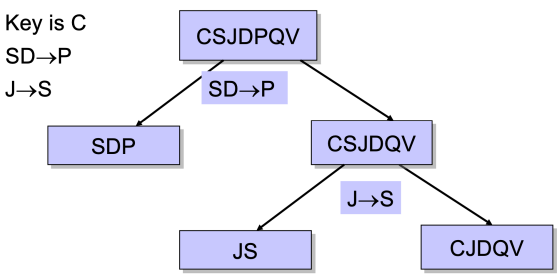
~ Suppose R is not in BCNF, A is an attribute where X → A is a FD violates the BCNF condition

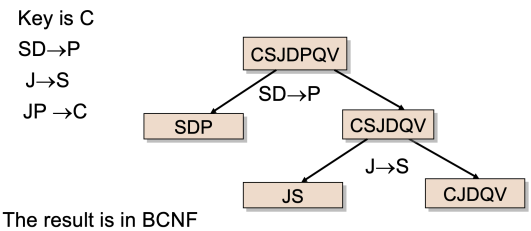
Step 1: Decompose R into XA and R-A

Repeat step 1 until all relations are BCNF

★ It is a lossless join decomposition

★ Not necessary dependency preserving





The result is in BCNF

Does not preserve JP→C, we can add a schema: CJP

⇒ Here is a redundancy of CJP

★ Not preserve dependency

• Canonical Cover

~ a minimal and equivalent set of functional dependency

equivalent: $E^+ = F^+$ if E is equivalent to F

EG. $R = (A, B, C)$

$F = \{A \to BC, B \to C, A \to B, AB \to C\}$

∵ $A \to BC$ implies $A \to B$ and $A \to C$ | Compute the closure of A. $A^+ = ABC$

∴ $A \to B$ is redundant

$F' = \{A \to BC, B \to C, AB \to C\}$

Compute $(AB)^+$ under F' ∵ $(AB)^+ = ABC$

∴ $AB \to C$ is redundant

$F'' = \{A \to BC, B \to C\}$

∵ From $A \to B$ and $B \to C$, we deduce $A \to C$ (transitivity)

∴ From $A \to B$ and $A \to C$, we get $A \to BC$ (union)

$F''' = \{A \to B, B \to C\}$

Canonical Cover F_c

1. Every functional dependency $\alpha \to \beta$ in F_c contains no extraneous attributes in α
2. Every functional dependency $\alpha \to \beta$ in F_c contains no extraneous attributes in β
3. No two dependencies that $\alpha_1 \to \beta_1$ and $\alpha_2 \to \beta_2$ in F_c such that $\alpha_1 = \alpha_2$

Step 1: Replace any $\alpha_1 \to \beta_1$ and $\alpha_1 \to \beta_2$ by $\alpha_1 \to \beta_1\beta_2$

Step 2: Delete any extraneous attributes from any $\alpha \to \beta$

Repeat until F does not change

- 3NF Synthesis Algorithm

for each $\alpha \rightarrow \beta$ in F_c

  if no schema in result contains $\alpha\beta$,

    then add schema $\alpha\beta$ to result

if no schema in result contains a candidate key for R,

  then choose any candidate key α for R,

      add schema α to the result

Ch6. Storage & Indexes

A. Disks and Files

~ DBMS stores information on hard disks

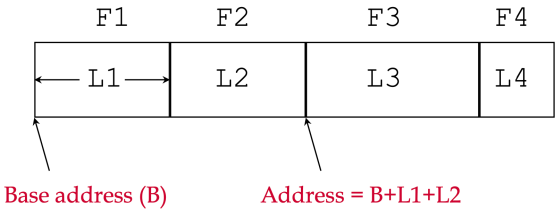
READ: transfer data from disk to main memory RAM

WRITE: transfer data from RAM to disk

~ Disk blocks or pages

page: fixed size and contains a sequence of records

• Record Format: Fixed Length



Ch7. Evaluating Relational Operators

A. Cost Calculations

Example

Schema: Sailors(***sid***: integer, ***sname***: string, ***rating***: integer, ***age***: real)
Reserves(***sid***: integer, ***bid***: integer, ***day***: dates, ***rname***: string)

Assumption: Sailors: 500 pages, 80 tuples/page, 50 bytes/tuple
Reserves: 1000 pages, 100 tupes/page, 40 bytes/tuple

Query
```
SELECT *
FROM Reserves R
WHERE R.rname = 'Joe'
```

1. No index, Unsorted (R contains M pages)

1. Scanning the entire R relation by reading the page one by one
2. For each page scanned, checking the condition on each tuple
3. If the condition is met, add the tuple to results

⇒ Cost = M I/Os

EG. Select rname = 'Joe' from R

⇒ Cost = 1000 I/Os

2. No index, R sorted on R.attr

1. Binary search on R.attr for value to locate the first tuple that satisfies the condition

~ cost = $\lg M$

2. Start from that position, scan the relation until the condition become untrue

~ cost depends on # tuples satisfying the condition (10% ⇒ 10% · M)

⇒ Cost = $\lg M + pM \quad \forall p \in [0,1]$

EG. Search 'Chan' < rname < 'Lin' and assume 10% of tuples satisfy the condition

⇒ Cost = $\lg 1000 + 10\% \cdot 1000 = 110$ I/Os

3. B+ Tree Index on R.attr

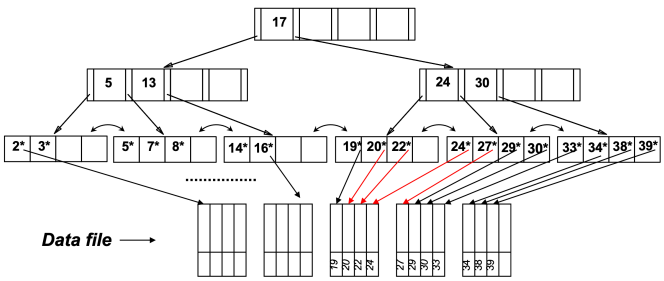
1. Search the tree to find the first data entry that points to a qualifying tuple of R
   - ~ cost = height of the tree + 1
2. Scan the leaf page to retrieve all the data entries in which the key value satisfy the condition
   - ~ cost depends on the number of such data entries
3. For each data entry retrieved, follow the pointer to get the corresponding tuple of R
   - ~ let N qualifying tuples and P tuples that can be stored in a page

   cost if index is clustered $\leq \lceil N/P \rceil + 1$

Cluster B+ Tree Index

EG. Search 'Chan' < rname < 'Lin' and assume 10% of tuples satisfy and level = 3

⇒ Cost = 3 + 1000 × 10% = 103 I/Os

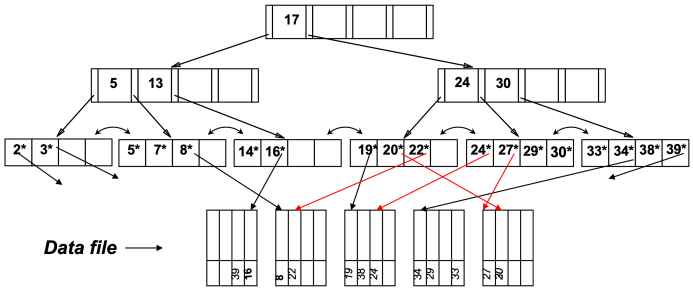


⇒ 3+2

Unclustered B+ Tree Index

EG. Assume the total number of pages is 1/10 of the number of data page

⇒ Cost = $3 + 1000 \times \frac{1}{10} \times \frac{1}{10} = 13$ I/Os



⇒ 3+2-1+4 = 8

4. Hash index on R.attr (op is =)

1. Calculate the hash value for value

   ~ Cost = 0

2. Get the directory entry identified by hash value

   ~ Cost = 0/1 1 if directory does not fit in the memory (not in main memory)

3. Retrieve the bucket page pointed by the directory entry

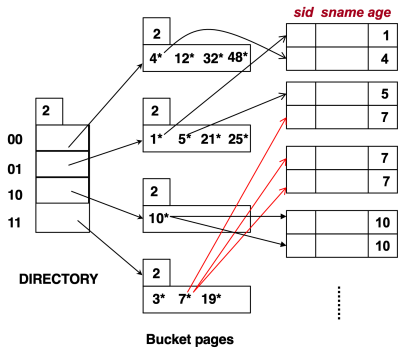
   ~ Cost = 1/2 (we may go through the overflow pages)

4. For each data entry in bucket, retrieve the qualifying tuple

   ~ Cost depends on # qualifying tuples and whether clustered index

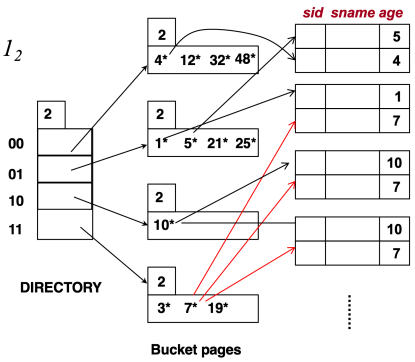
Clustered Index

- Selection Condition: age = 7
- **h**(7) = *7 mod 32 = 7 = 111*$_2$
- Cost
  - Calculate **h**(7): 0
  - Get directory entry: 0 (assume directory is in main memory)
  - Get bucket page: 1
  - Get qualifying tuples: 2



Unclustered index

- Sel. Cond. age = 7
- **h**(7) = *7 mod 32 = 7 = 111*$_2$
- Cost
  - Get directory entry: 0 (assume directory is in main memory)
  - Get bucket page: 1
  - Get qualifying tuple: 3



$2 \times M \times \lceil \log_{B-1} \frac{M}{B} \rceil + 1$ M pages B buffers.

B. Projection

~ General Method:

1. Scan relation R and discard the unwanted attributes

2. Eliminate duplicates

~ Projection based on sorting (R - 1000 pages, T - 250 pages, 20 buffer available)

1. Scan R, write the target attributes of each tuple to a temporary file T

   1000 + 250 = 1250 I/Os

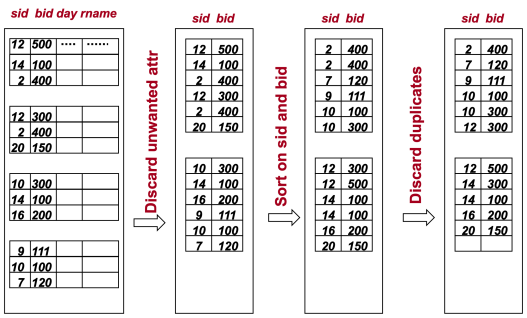
2. Sort T

   $2 \times 250 \times (\lceil \log_{19} \frac{250}{19} \rceil + 1) = 1000$ I/Os

3. Scan the sorted file, compare the adjacent tuples and discard duplicates

   250 I/Os

Example: $\Pi_{sid, bid}$(Reserves) based on sorting



C. Join

- Block Nested Loops Join

~ Suppose we have enough buffers to hold B+2 pages, then we can read B pages at a time

```
For each block P of Reserves
    for each page q of Sailors
        for each r ∈ P and s ∈ q such that r.sid=s.sid
        add <r, s> to the result
```

```
SELECT *                        (B = 100)
FROM Reserves R, Sailors S      (R has 1000 pages, 100 tuples/page)
WHERE R.sid = S.sid             (S has 500 pages, 80 tuples/page)
```

1. No index

∵ B = 100 ⇒ Reserves can hold $\frac{1000}{100}$ = 10 blocks

For each block, scan Sailors for 500 I/Os

∴ Total Cost = 1000 + 10x500 = 6000 I/Os

⇄ If we swap them, Total Cost = 500 + $\frac{500}{100}$ x1000 = 5500 I/Os

2. Hash index

Case 1 : Assume an index on sid of the Sailors relation

1. Scan the Reserves relation 1000
2. For each tuple in Reserves, an average of 1.2 to get the bucket page containing the matching Sailors entry 1.2
3. Total # Reserves entries 1000 x 100
4. # Reserves entries = 100000 while # Sailors = 40000 ⇒ each reservations = 0.4 sailors

∴ Total Cost = 1000 + 1000x100x(1+1.2) = 221000 I/Os

Case 2 : Clustered Index. Assume an index on sid of the Reserves relation

4. # Reserves entries = 100000 while # Sailors = 40000 ⇒ each sailors = 2.5 reservations

Clustered Index ⇒ 2.5 reservations likely on the same page

∴ Total Cost = 500 + 500x80x(1+1.2) = 88500 I/Os

Case 3: Unclustered Index

Clustered Index ⇒ 2.5 reservations likely NOT on the same page

∴ Total Cost = 500 + 500x80x(2.5+1.2) = 148500 I/Os

D. Sort - Merge Join

~ Cost for joining two sorted relations are M + N

Ch8. Database Concurrency Control

A. Transition

Transition ~ a sequence of read / write operations a set of transitions $T = \{T_1, T_2, \ldots, T_n\}$

T1
Read[a]
Write[b]

Transaction T_1

T2
Read[b]
Write[a]

Transaction T_2

History ~ a sequence of operations in $T_1, T_2, \ldots, T_n$ $\forall i$

1. each operation in T_i appears exactly once
2. operations from T_i appear in the same order as in T_i

Assume: Each T_i is correct when executed individually

T1	T2
Write[b]	
	Read[b]
Read[a]	
	Write[a]

Order of operations is wrong

Invalid Schedule

T1	T2
Read[a]	
Write[b]	
	Read[b]
	Write[a]
Read[a]	

duplicated operations

Invalid Schedule

T1	T2
Read[a]	
	Read[b]
Write[b]	
	Write[a]

Valid schedule (interleaved schedule)

$r_1[a]\ r_2[b]\ w_1[b]\ w_2[a]$

T1	T2
Read[a]	
Write[b]	
	Read[b]
	Write[a]

Valid schedule (serial schedule)

$r_1[a]\ w_1[b]\ r_2[b]\ w_2[a]$

- Serial Schedule (Accept Schedule)

~ a schedule is serial if $\forall i, j$, either

- all the operations in T_i appear before all the operations in T_j OR
- all the operations in T_j appear before all the operations in T_i

- Equivalent

~ same effect:

the effects of schedule are the value produced by the White operations of unaborted transactions

Assume that if each transaction's Reads read the same value in two schedules,

then all Writes write the same value in both schedules

If each data item x, the final Write on x is the same in both schedules,

then the final values of all data items will be the same in both schedules

B. View Serializability

- View Equivalent

  ~ Two Schedules H and H' are view equivalent if

  1. they are over the same set of transactions and have the same operations AND
  2. for any unaborted T_i, T_j and for any x, if T_i reads x from T_j in H, then T_i reads x from T_j in H' AND
  3. for each x, if $w_i[x]$ is the final write of x in H, then it is also the final write of x in H'

- View Serializable VSR

Assume: there is a transaction T_b that initializes the value for all data objects

~ a schedule is view serializability if it is view equivalent to a serial schedule

$r_3[x]w_4[x]w_3[x]w_6[x]$
- T_3 read x from T_b.
- The final write for x is $w_6[x]$.
- View equivalent to T_3 T_4 T_6.

$r_3[x]$ $w_4[x]$ $w_3[x]$
- T_3 read x from T_b.
- The final write for x is $w_3[x]$.
- Not serializable.

$r_3[x]$ $w_4[x]$ $r_7[x]$ $w_3[x]$ $w_7[x]$
- T_3 read x from T_b.
- T_7 read x from T_4.
- The final write for x is $w_7[x]$.
- View equivalent to T_3 T_4 T_7.

$w_1[x]$ $r_2[x]$ $w_2[x]$ $r_1[x]$
- T_2 read x from T_1.
- T_1 read x from T_2
- The final write for x is $w_2[x]$.
- Not serializable.

C. Conflict Serializability

~ Non-Conflicting Operations: can always be swapped without affecting the result of the other operations

ie. $op_i(A)\ op_j(B) \leftrightarrow op_j(B) op_i(A)$ if $A \neq B$

★ Conflict Operations:

1. $r_i[x]\, r_j[x] \leftrightarrow r_j[x] r_i[x]$
2. $r_i[x]\, w_j[x]$
3. $w_i[x]\, r_j[x]$
4. $w_i[x]\, w_j[x]$

} Cannot Swap

Two operations are conflicting with each other if they both operate on the same data object and at least one of them is a Write

- Conflict Equivalent

~ Two histories H and H' are conflict equivalent if

1. they are defined over the same set of transactions and have the same operations
2. they order conflicting operations in the same way

- Conflict Serializable CSR

~ a history is conflict serializable if it is conflict equivalent to a serial history

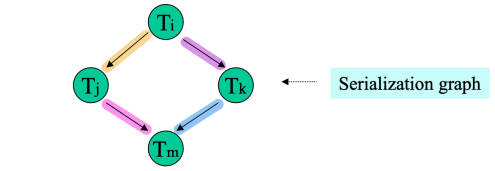
D. Serialization Graph

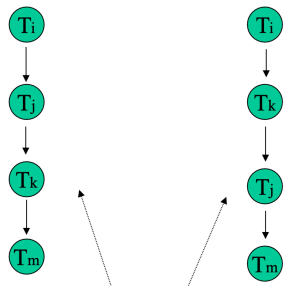
Recall the Conflict Operations:

2. $r_i[x]\ w_j[x]$
3. $w_i[x]\ r_j[x]$
4. $w_i[x]\ w_j[x]$

} Cannot Swap $T_i \rightarrow T_j$

$w_i[a]\ r_i[b]\ r_k[a]\ w_j[b]\ w_j[d]\ w_k[c]\ r_m[c]\ r_m[d]$



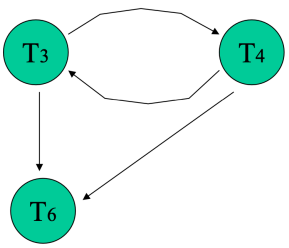


Two possible serialization orders

☆ NOT Conflict Serializable

~ if there is a cycle in the serialization graph

$r_3[x]\ w_4[x]\ w_3[x]\ w_6[x]$



☆ Conflict serialization ⊆ View serialization

E. Lock-based Protocols

1. Shared-Locks (read operation)

2. Exclusive-Locks (write operation)

Ch9. Recovery

A. Recoverability

- Recoverable RC

~ transitions may be aborted due to logical failure (deadlock)

~ recoverability is required to ensure that aborting a transaction does not change the semantics of committed transaction's operations.

EG. $w_1[x]\, r_2[x]\, w_2[y]\, c_2\, c_1$ ⇒ Not recoverable: T_2 has committed before T_1

~ H is RC if T_i reads from T_j, then $c_j < c_i$

ie. if each transaction commits after the commitment of all transitions from which it reads

- Avoiding Casading Aborts

~ even for recoverable execution, aborting a transaction may trigger further abortions

~ H is ACA if T_i reads x from T_j, $c_j < r_i[x]$

ie. a transition may read only those values that are written by committed transitions or itself

- Strict Execution

~ ACA is not always enough from the partical point of view