RDBMS and SQL

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Lecture 11, 11 November 2026

Query processing

- Translate the query from SQL into relational algebra
- Evaluate the relational algebra expression
- Challenges
  - Many equivalent relational algebra expressions

    $\sigma_{salary<75000}(\pi_{salary}(instructor))$ vs $\pi_{salary}(\sigma_{salary<75000}(instructor))$
  - Many ways to evaluate a given expression
- Query plan
  - Annotate the expression with a detailed evaluation strategy key values
    - Use index on *salary* to find instructors with $salary < 75000$
    - Or, scan entire relation, discard rows with $salary \geq 75000$

Query optimization

- Choose plan with lowest cost
- Maintain database catalogue — number of tuples in each relationn, size of tuples, ...
- Assess cost in terms of disk access and transfer, CPU time, ...
- For simplicity, ignore in-memory costs (CPU time), restrict to disk access
- Disk accesses
  - Relation r occupies b_r blocks
  - Disk seeks — time t_S per seek
  - Block transfers — time t_T per transfer

Cost of query processing

- Selecting rows, depending on key, index etc
- External merge sort
- Processing joins
  - Nested loop join
  - Block nested loop join
  - Merge join
  - Hash join

$\sigma_\theta(r)$

Query optimization

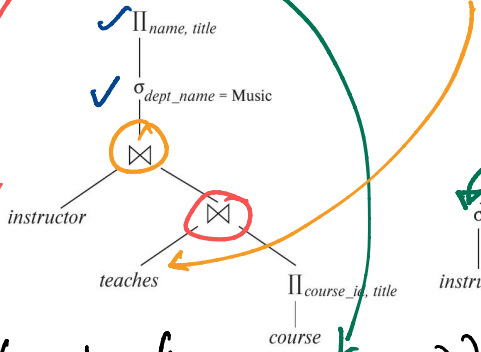
- Choose plan with lowest cost

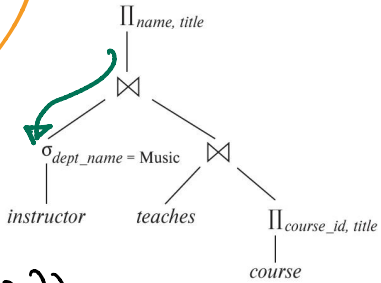
Query optimization

- Choose plan with lowest cost
- *Find names and course titles of courses taught by instructors from Music Dept*

Query optimization

- Choose plan with lowest cost
- *Find names and course titles of courses taught by instructors from Music Dept*





$\Pi(\sigma(\text{inst} \bowtie (\text{teach} \bowtie \text{course})))$

Transforming expressions

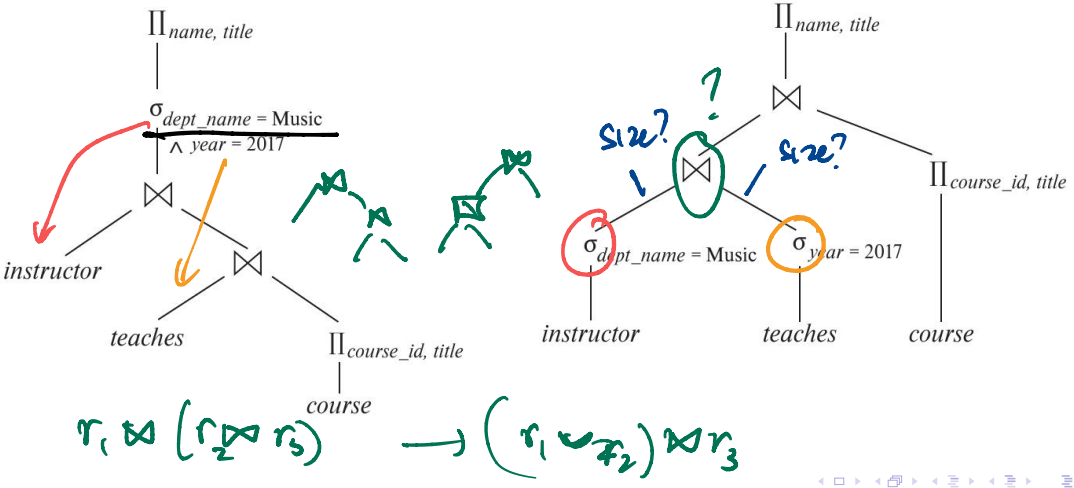
$$\sigma_{\theta_1}(\sigma_{\theta_2}(r)) = \sigma_{\theta_2}(\sigma_{\theta_1}(r)) = \sigma_{\theta_1 \wedge \theta_2}(r)$$

$$r_1 \bowtie r_2 \bowtie r_3 = (r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

"Rewriting" rules

$$\sigma_\theta(r_1 \bowtie r_2) \rightsquigarrow \sigma_\theta(r_1) \bowtie r_2$$ Depends on θ, ...

Transforming expressions



Maintaining a database catalogue

- n_r — number of tuples in r
- b_r — number of blocks used by r
- ℓ_r — size of a tuple in r
- f_r — blocking factor of r, how many tuples fit in a block
- $V(A, r)$ — number of distinct values of attribute A in r
  - Store distribution of values as histogram

Estimate

Cost of operation
Depends on input size & "type"

Size of output

$$\pi_c(\sigma_\theta(r))$$

Estimating output of an operation

- Selection
  - Simple, range, conjunction, disjunction

$V(A, r)$

Estimating output of an operation

- Selection
  - Simple, range, conjunction, disjunction
- Join
  - Keys and non-keys



lossless join

$(r_1 \cap r_2) \rightarrow r_1$ OR

$(r_1 \cap r_2) \rightarrow r_2$

Estimating output of an operation

- Selection
  - Simple, range, conjunction, disjunction
- Join
  - Keys and non-keys
- Projection

Estimating output of an operation

- Selection
  - Simple, range, conjunction, disjunction
- Join
  - Keys and non-keys
- Projection
- Aggregation

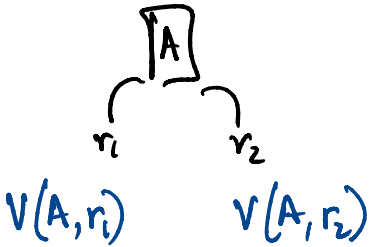
$\#\ groups = V(A, r)$

Estimating output of an operation

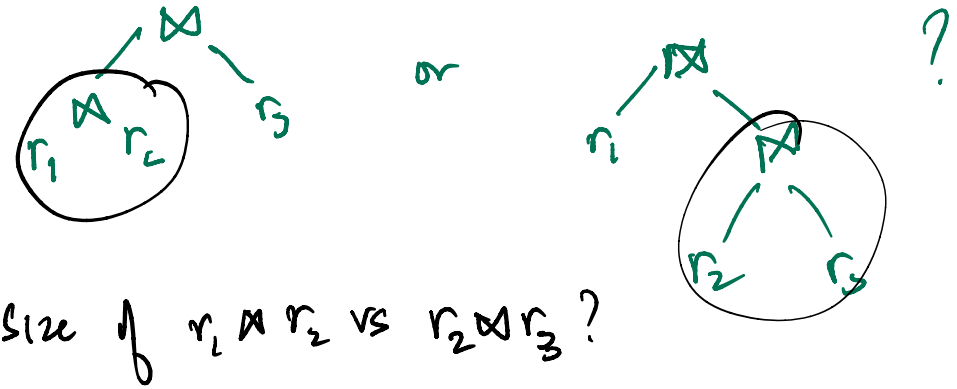
- Selection
  - Simple, range, conjunction, disjunction
- Join
  - Keys and non-keys
- Projection
- Aggregation
- Set operations

Estimating output of an operation

- Selection
  - Simple, range, conjunction, disjunction
- Join
  - Keys and non-keys
- Projection
- Aggregation
- Set operations
- Outer joins



Join ordering



Join ordering

Related problem

Matrix multiplication

$1 \times 100 \quad 100 \times 1 \quad 1 \times 100$

$M_1 \times M_2 \times M_3$

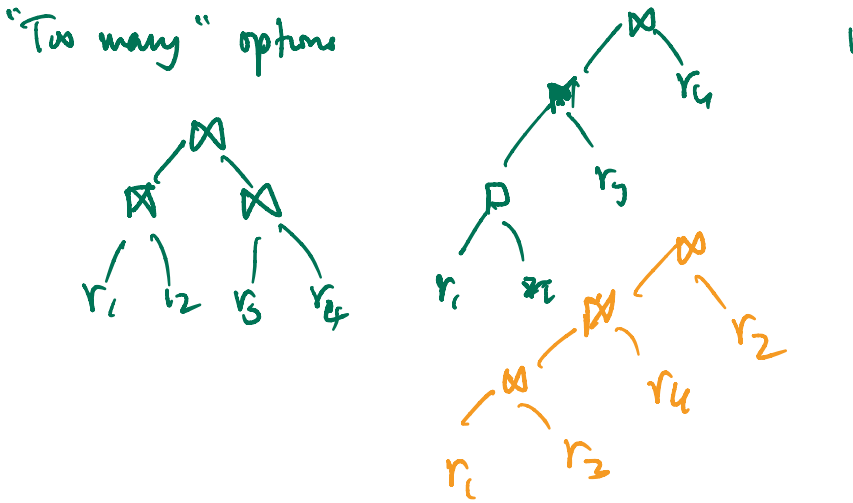
$(M_1 \times M_2) \times M_3 \rightarrow 1 \times 100$

$1 \times 1 \quad 1 \times 100$

$M_1 \times (M_2 \times M_3)$

$1 \times 100 \quad 100 \times 100$

Join ordering



Heuristics

- Perform selection early

Heuristics

- Perform selection early
- Perform projection early

Heuristics

- Perform selection early
- Perform projection early
- Perform most restrictive selection/join first

"Input is known in advance"

Transactions

All or nothing set of operations

Desirable properties

- Atomicity – Indivisible operation

Desirable properties

- Atomicity
- Consistency – Never expose an inconsistent state

Desirable properties

- Atomicity
- Consistency
- Isolation

A transaction should not "know about" parallel transactions

Desirable properties

- Atomicity
- Consistency
- Isolation
- Durability

ACID properties

Desirable properties

- Atomicity
- Consistency
- Isolation
- Durability
- ACID properties

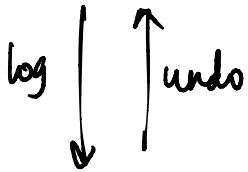
Concurrency

Multiple conflicting transactions in parallel

Entire transaction is known in advance

Transaction logs

- Log each update before it happens
- Rollback updates in case of failure



Concurrent execution and schedules

Transfer 50 from A to B

T_1: read(A);
$A := A - 50$;
write(A);
read(B);
$B := B + 50$;
write(B).

Transfer 10% of A to B

T_2: read(A);
$temp := A * 0.1$;
$A := A - temp$;
write(A);
read(B);
$B := B + temp$;
write(B).

Concurrent execution and schedules

A = B = 1000

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit

950

1050

950 − 95 = 855

1145

Concurrent execution and schedules

Schedule

Order in which operations are performed

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	

900

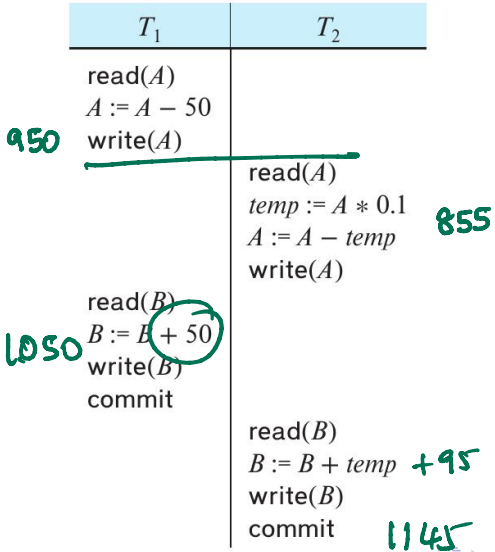
850

1100

1150

Concurrent execution and schedules

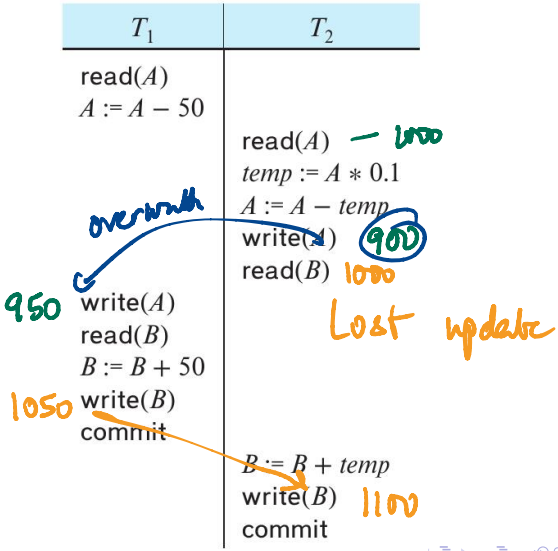
Concurrent schedules



Concurrent execution and schedules

"Bad" concurrent schedule

What is the definition of a "good" concurrent schedule



Serializability

- Serial schedule — each transaction executes as a block, no interleaving

Serializability

- Serial schedule — each transaction executes as a block, no interleaving
- Serializable schedule — equivalent to *some* serial schedule

?

Serializability

- Serial schedule — each transaction executes as a block, no interleaving
- Serializable schedule — equivalent to *some* serial schedule
- Conflicting operations — two operations on the *same* value where *at least one is a write*

Serializability

- Serial schedule — each transaction executes as a block, no interleaving
- Serializable schedule — equivalent to *some* serial schedule
- Conflicting operations — two operations on the *same* value where *at least one is a write*
- Conflict equivalence — one schedule can be transformed into the other by reordering non-conflicting operations

Swap operations to transform serializable to serial

Serializability

- Serial schedule — each transaction executes as a block, no interleaving
- Serializable schedule — equivalent to *some* serial schedule
- Conflicting operations — two operations on the *same* value where *at least one is a write*
- Conflict equivalence — one schedule can be transformed into the other by reordering non-conflicting operations
- Conflict serializable — can be reordered to a conflict-equivalent serial schedule

Conflict equivalence

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit

≈

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(B)
	$B := B + temp$
	write(B)
	commit