OPTIMIZATION OF RELATIONAL ALGEBRA

--> Do Selection first, then project, only join when you have to!

① JOIN V. CART. PRODUCT
- joins are better

$R \bowtie_c S = \sigma_c(R \times S)$

② PUSH SELECTION DOWN
⇒ DO SELECTION AS EARLY AS POSSIBLE
- it reduces the size of the data

$\sigma_c(\pi_L(R)) \longleftrightarrow \pi_L(\sigma_c(R))$

$\sigma_{R.x=5}(R \bowtie_{R.a=S.b} S) \longleftrightarrow \sigma_{R.x=5}(R) \bowtie_{R.a=S.b} S$

③ AVOID UNNESSARY JOINS

find customers having account balances below 100 and loans above 10000

$R1 \leftarrow \pi_{custName}(Depositor \bowtie \pi_{accountNum}(\sigma_{balance<100}(Account)))$
$R2 \leftarrow \pi_{custName}(Borrower \bowtie \pi_{loanNum}(\sigma_{amount>10000}(Loan)))$
$Result \leftarrow R1 \cap R2$

⇒ Much better than trying to join 4 tables!

RELATIONSHIPS AMONG OPERATORS

① JOIN ↔ CARTESIAN PRODUCT + SELECT
$R \bowtie_c S = \sigma_c(R \times S)$

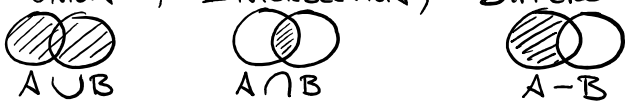
② SELECTION is Commutative
$\sigma_{c2}(\sigma_{c1}(R)) = \sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c1 \text{ AND } c2}(R)$

③ ORDER BETWEEN SELECTION & PROJECTION
$\sigma_c(\pi_L(R)) \rightarrow \pi_L(\sigma_c(R))$
$\pi_L(\sigma_c(R)) \rightarrow \sigma_c(\pi_L(R))$ * ONLY if L has all cols needed for C

④ JOIN IS COMMUTATIVE
$R \bowtie_c S = S \bowtie_c R$

⑤ ORDER BETWEEN SELECTION & JOIN
$\sigma_{c1}(R \bowtie_{c2} S) \rightarrow (\sigma_{c1}(R) \bowtie_{c2} S)$

① Set operations: UNION, INTERSECTION, DIFFERENCE



② PROJECTION $\pi_L(R)$: SELECT clause – chooses some columns to use
L is a list consisting of col name, col renames (eg A as B), or expressions (eg A+B as Z)

③ SELECTION $\sigma_C(R)$: WHERE clause – choose some tuples to use
C is a conditional to apply to each tuple in R

③ COMBINING TABLES: FROM clause – choose which tables to use & how to join

① CROSS-PRODUCT	$A \times B$:	pairs each $a \in A$ with each $b \in B$	⇒ makes huge tables
② NATURAL JOIN	$A \bowtie B$:	pairs each matching attribute in matching columns	
③ THETA JOIN	$A \bowtie_c B$:	pairs each $a \in A$ with each $b \in B$ if $C(a,b)$ holds true	
④ OUTER JOIN	$A \overset{\bowtie}{}_c B$:	pairs according to theta join, then pads the dangling tuples with $\perp$	

A:

X	Y
0	1
1	0

B:

X	C
1	0
1	1

$A \times B$:

AX	AY	BX	BC
0	1	1	0
1	0	1	0
0	1	1	1
1	0	1	1

$A \bowtie B$:

X	Y	C
1	0	0
1	0	1

$A \bowtie_{Y=C} B$

AX	AY	BX	BC
0	1	1	1

$A \overset{\bowtie}{}_{Y=C} B$:

X	Y	C
1	0	0
1	0	1
0	1	$\perp$

④ RENAMING: $\rho_{S(A_1, A_2 \ldots, A_n)}(R)$: AS operator

⑤ DUPLICATE ELIMINATION: $\delta(R)$: DISTINCT operator – returns R with one copy of each tuple in R
⇒ Turns a bag into a set

⑥ SORTING: $\tau_L(R)$: ORDER BY CLAUSE
- L is a list of fields to sort by, where ties are broken by fields later in the list

⑦ AGGREGATION & GROUPING: $\gamma_L(R)$: GROUP BY clause
- L is a list of GROUPING ATTRIBUTES (attributes $\in R$) and AGGREGATED ATTRIBUTES (operator applied to cols of R)
  $\gamma_{activity, SUM(hours) \rightarrow timespent}(HoursLog)$
- executing $\gamma_L(R)$: ① Partition R into groups, where each group has tuples with a distinct assignment of the grouping attributes
  → if no grouping attrs specified, R is one group
  ② for each group, produce one tuple consisting of:
  - the grouping attributes for that group
  - the aggregations over the tuples in that group