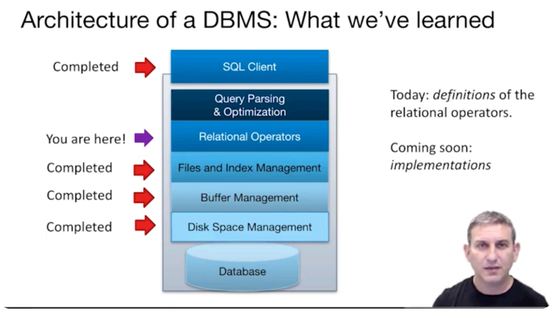
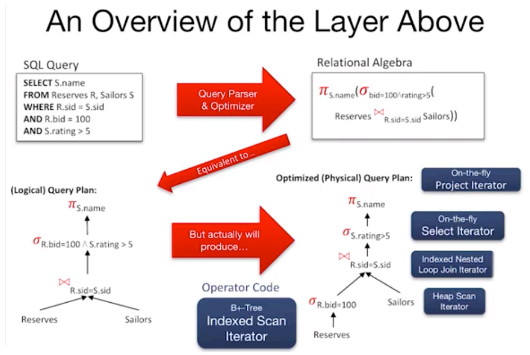
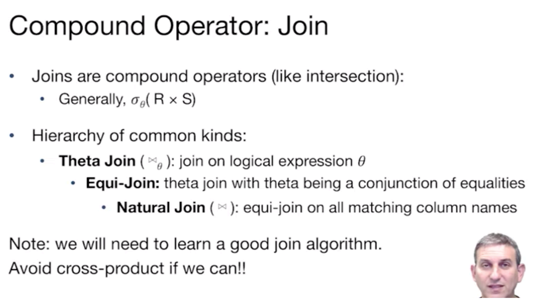
- Optimizer works on operations in relational algebra language which can be represented as a logical query plan (tree shaped), basically a strategy to execute the query
  - optimizer produces an optimized physical query plan which generates an algorithm for every operator in the tree to pump data through tree via operations called iterators (more on all this later)
  - SQL - declarative expression of the query result (what you want)
    - the query optimizer figures out how to get it
  - Relational algebra - operational description of a computation; it's an ordered plan
    - algebra on sets
    - operational description of transformations of sets
    - Codd's Theorem: equivalence in expressivity b/t relational calculus (essentially a SQL-like lang that describes result of computation based on first order logic) and relational algebra
      - connects declarative representation of queries with operational description (ask and computer will deliver somehow)
      - can compile SQL into relational algebra
    - specify explicit order of operations, uses closed set of operators


Architecture of a DBMS: What we've learned
Completed
SQL Client
Query Parsing & Optimization
You are here!
Relational Operators
Completed
Files and Index Management
Completed
Buffer Management
Completed
Disk Space Management
Database
Today: *definitions* of the relational operators.
Coming soon: *implementations*


An Overview of the Layer Above
SQL Query
SELECT S.name
FROM Reserves R, Sailors S
WHERE R.sid = S.sid
AND R.bid = 100
AND S.rating > 5
Query Parser & Optimizer
Relational Algebra
$\pi_{S.name}(\sigma_{bid=100 \wedge rating>5}($ Reserves $\bowtie_{R.sid=S.sid}$ Sailors$))$
Equivalent to...
(Logical) Query Plan:
$\pi_{S.name}$
$\sigma_{R.bid=100 \wedge S.rating > 5}$
$\bowtie_{R.sid=S.sid}$
Reserves
Sailors
But actually will produce...
Operator Code
B+-Tree Indexed Scan Iterator
Optimized (Physical) Query Plan:
On-the-fly Project Iterator
$\pi_{S.name}$
On-the-fly Select Iterator
$\sigma_{S.rating>5}$
Indexed Nested Loop Join Iterator
$\bowtie_{R.sid=S.sid}$
Heap Scan Iterator
$\sigma_{R.bid=100}$
Sailors
Reserves

- Set operators
  - Union (U):
    - 2 input relations must be compatible, i.e. same # fields (columns), fields in corresponding positions have same *type*
    - SQL expression: UNION
      - UNION ALL doesn't remove duplicates in the union
  - Set difference (-):
    - both input relations must be compatible, returns tuples in R1 but not in R2
    - SQL expression: EXCEPT
      - EXCEPT ALL does not remove duplicates of matches
  - Cross-Product (x): R1 x S1
    - each row of R1 paired with each row of S1
    - R1 and S1 don't need to be schema compatible
  - Renaming (rho):
    - renames relations and their attributes
    - relational algebra doesn't require names (just use positional args) but names are useful
    - rho(<output relation>(<renaming list>), <input relation>)
      - first arg schema change definition
        - output relation is name of the new table
        - renaming list is <pos -> new name> pairs saying: rename column at <pos> the name <new name>
  - **Selection** (sigma) selects based on some conditions theta (i.e. a WHERE clause)
    - choose desired rows
    - e.g. sigma_{rating > 8}(S)
    - corresponds to WHERE clause in SQL
    - duplicate elimination is not needed since the input is a set of tuples (no dups) and we only removed some tuples



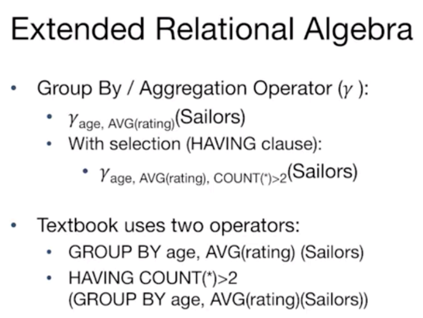
Compound Operator: Join

- Joins are compound operators (like intersection):
  - Generally, σ_θ(R × S)
- Hierarchy of common kinds:
  - **Theta Join** ($\bowtie_\theta$): join on logical expression θ
    - **Equi-Join:** theta join with theta being a conjunction of equalities
      - **Natural Join** ($\bowtie$): equi-join on all matching column names

Note: we will need to learn a good join algorithm.
Avoid cross-product if we can!!

  - **Projection** (pi_{expression}) projects a table onto a different schema (same # rows) given by the expression (e.g. prune # columns)
    - choose desired columns
    - corresponds to the select list in SQL
    - e.g. pi_{name, age}(S)

            - remember that all duplicate values are removed
    - Compound operators:
        - intersection (INTERSECT)
            - both input relations must be compatible
            - basically S1 && S2 = S1 - (S1 - S2)
                - just the stuff that's not different
        - join (JOIN)
            - does a cross product between tables
            - theta join: e.g. join on sid = sid
                - e.g. join on age1 < age2
            - natural join: joins on all columns with matching names (join on matching values for these cols), project away the 2 matching columns into 1 column (common for foreign key joins)
    - Extended relational algebra
        - group by / aggregation operator: gamma_{expression}
            - expression is a group by column list (which can include aggregation function over input columns) + having condition
- The algebra is closed, meaning each input/output is a relation
    - everything is typed, i.e. has a set type
    - uses set semantics, meaning NO DUPLICATES (i.e. no duplicate tuples) in relations
        - different from SQL which uses multiset (bag) semantics



Extended Relational Algebra

- Group By / Aggregation Operator (γ):
    - $\gamma_{\text{age, AVG(rating)}}$(Sailors)
    - With selection (HAVING clause):
        - $\gamma_{\text{age, AVG(rating), COUNT(*)>2}}$(Sailors)
- Textbook uses two operators:
    - GROUP BY age, AVG(rating) (Sailors)
    - HAVING COUNT(*)>2
      (GROUP BY age, AVG(rating)(Sailors))