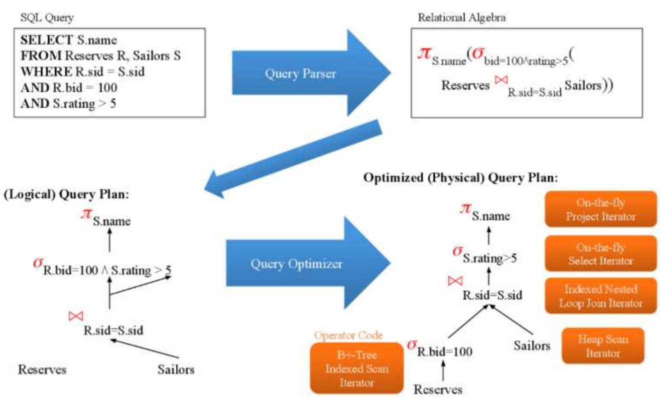
CS186 Midterm 2 Cheat Sheet

Index I on primary key matches selection: Height(I) + 1
Clustered on one or more: (NPages(I) + NPages(R)) * product of RF's of matching selects
Non-clustered index I matching one or more selections: (NPages(I) + NTuples®) * product of RF's of matching selects
Sequential scan: NPages(R)

Sorting and Hashing:

- Sorting **# of passes** = 1 + ceil($\log_{B-1}$(ciel(N/B)))
- Sorting **cost**: 2N * # of passes
- Hashing cost: 2N * #passes = 4N (assuming no overflow)
  - Divide (partitions to B-1)

Relational Algebra:



- **Codd's theorem** established equivalence in expressivity between **relational calculus** and **relational algebra**
- Unary Operators:
  - **Projection (π)**: Retains only the desired columns (vertical)
  - **Selection (σ)**: Selects a subset of rows (horizontal) (WHERE)
  - **Renaming (ρ)**: Rename attributes and relations

$$\varrho(\underbrace{Temp1}_{\text{Output Relation Name}}(\underbrace{1 \rightarrow sid1, 4 \rightarrow sid2}_{\text{Renaming List position} \rightarrow \text{New Name}}), \underbrace{R1 \times S1}_{\text{Input Relation}})$$

- Binary Operators (on pairs of relations):
  - **Union (∪):** Tuples in r1 or in r2 (removes duplicates)
  - **Set-Difference (—):** Tuples in r1, but not in r2 (EXCEPT)
  - **Cross-Product (×):** Allows us to combine two relations.
- Compound Operators (common "macros" for the above):
  - **Intersection (∩):** Tuples in r1 and in r2
  - **Joins ($\bowtie_\theta$, ⋈):** Combine relations that satisfy predicates
- **Division operator:** all entries **x** in **A** such that for all **y** in **B** there is an **(x, y)** in **A.** A / B = $\pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$
- **Group By/Aggregation** operator (γ): (HAVING) $\gamma_{age,\ AVG(rating),\ COUNT(*)>2}$(Sailors) or GROUP BY

Iterators and Joins

- Cost Notation
  - [R]: the number of pages to store R
  - p_R: number of records per page of R
  - |R|: the cardinality (number of records) of R
    - |R| = pR*[R]
- **SNLJ**: foreach **record** r in R do for each **record** s in S do join
  - Cost: (p_R *[R]) * [S] + [R] (works for more than equality)
- **PNLJ**: Assume we use a small amount of RAM (to store a page of R and a page of S)
  - Foreach **page** b_R in R do: Foreach **page** b_S in S do:
  - Foreach **record** r in b_R do: Foreach **record** s in b_S do: join
  - Cost: [R] * [S] + [R]
- **BNLJ**: Assume we use a B RAM (to store a page of R and a page of S)
  - Foreach **block** b_R of B-2 pages in R do:Foreach **page** b_S in S do:
  - Foreach **record** r in b_R do: Foreach **record** s in b_S do: join
  - Cost: ceil([R]/(B-2)) * [S] + [R]
- **Index Nested Loops Join**: uses index lookups (on one of the inputs)
  - Foreach tuple r in R do find matching S tuples: join
  - Cost = [R] + ([R] * p_R) * cost to find matching S tuples
- **Sort-Merge Join:** requires equality predicate (**equi-joins and natural joins**) 2 steps: **sort** and then **merge**
  - Cost: Sort R + Sort S + ([R] + [S]) (worst cast [R] *[S])

```
while not done {
  while (r < s) { advance r }
  while (r > s) { advance s }
  // assert r == s
  mark s // save start of "block"
  while (r == s) {
    // Outer loop over r
    while (r == s) {
      // Inner loop over s
      yield <r, s>
      advance s
    }
    reset s to mark
    advance r
  }
}
```

- **Simple Hash Join:**
  - Same as INLJ on hash
  - Simple: R fits in memory
- **Grace Hash join:** requires equality predicate: **divide** and **conquer**
  - Partition tuples from R and S by join key (2([R] +[S]) I/Os)
  - Matching: [R] + [S]

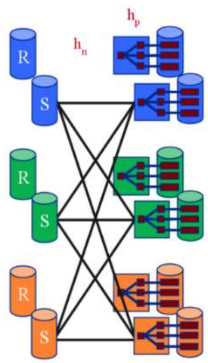
Armstrong's Axioms: Reflexivity, augmentation, transitivity, union, decomposition.
Attribute closure: all possibilities
1st normal form, all attribute atomic. **BCNF** X->A in F+ if x is a superkey of R and A in X.

Parallel Query Processing:

- **Speed-up**: fixed workload, want linear performance to parallelism
- **Scale-up**: increase workload, want constant performance
- **Pipeline** parallelism and **partition** parallelism
- Architectures: shared **memory &,** shared **disk,** shared **nothing**
- Query parallelism:
  - **Inter-query** (parallelism across queries). 1 query/processor
    - Careful of concurrency control (easy)
  - **Intra-query** (between operators within a single query)
    - **Inter-operator** can allow for pipeline parallelism

- $|f(x)|$ $|g(f(x)|$ $|h(g(f(x)))|$
- **Bushy** tree parallelism (left and right branch runs independently on 2 different branches)
- **Intra-operator** allows for between operators
- **Partition parallelism** stuff is partitioned to different machines

- **Data partitioning** is how we split data up across disks/machines
  - **Range:** like histograms. A-E in 1 bucket… (good for equijoins, range queries, group-by)
  - **Hash:** Each machine takes in the hash (equijoins/group-by)
  - **Round Robin:** As it implies (equally spaced load spreading)
- **Parallel hashing** is dead simple, add a step before first hash. Hash data to however many computers we plan to parallelize.
- **Parallel Grace Hash Join:**
  - Also dead simple. Perfect scale/speedup



- **Parallel sorting**: partition data by range (sample data) and data is sorted over 3 machines
- **Parallel sort merge join:** pass 0 … n-1 are like parallel sorting. Range partition R & S + before
- **Join shuffling:**
  - **Asymmetric shuffle** is just when R is already partitioned
  - **Broadcast join** take a small S, and copy it to all of the R machines and join there

Relational Query Optimization:

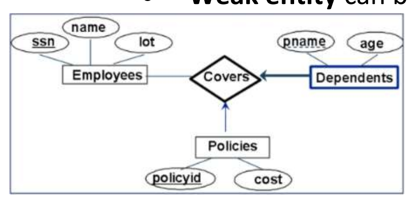
- **Query parser** checks correctness, authorization, generates parse tree
- **Query rewriter** converts queries to canonical form.
- **Plan space:** for a given query, what plans are considered
- **Cost estimation:** how is the cost of plan estimated
- **Search strategy:** how do we "search" in the "plan space?"
- Algebra equivalences: $\sigma_{c1\wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots(\sigma_{cn}(R))\ldots)$ (cascade)
- $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (commute) || $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{a1, \ldots, an-1}(R))\ldots)$ (cascade) || $R \times (S \times T) \equiv (R \times S) \times T$ (associative) || $R \times S \equiv S \times R$ (commutative)
- Apply **selections**, **projects** ASAP and avoid **Cartesian** products
- System R/Selinger optimizers is what we'll do. Cascades is other
  - Works well for 10-15 joins
  - Plan space: avoid overpriced subtree. Too large, must be pruned, only left-deep plans (avoid cartesion products)
  - Cost estimation: inexact, but works ok
  - Search algorithm: dynamic programming
- **Eager projection** you can project anything not needed later
- $\sigma_{R.a=S.b}(R \times S) \equiv R \bowtie_{R.a=S.b} S$ || $\sigma_{(R \bowtie S)} \equiv \sigma_{(R)} \bowtie S$
- Common physical properties of an output: sort order, hash grouping
- Index scan (sorted result), sort (sorted result), hash (grouped result)
- MergeJoin (input sorted, output sorted), NLJ (preserves sort order)
- System R: cost = #I/O + CPU-factor * #tuples
- Selectivity = |output| / |input| = RF
- Size estimation (1/10 default):
  - Col=value (1/NKeys)
  - Col1=col2 (1/max(NKeys(1), NKeys(2)))
  - Col > value (high(I)-value)/(High(I)-Low(I) +1)

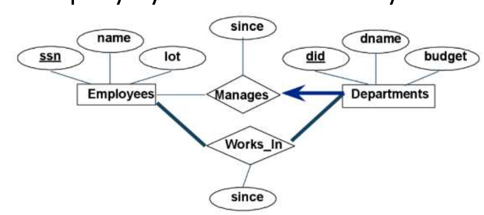
Statistic	Meaning
NTuples	# of tuples in a table (cardinality)
NPages	# of disk pages in a table
Low/High	min/max value in a column
Nkeys	# of distinct values in a column
IHeight	the height of an index
INPages	# of disk pages in an index

- Independence assumption: $\sigma_{p > 89} \wedge \sigma_{age < 26}$? 50% * 46% = 23%
- $\sigma_{p > 89} \vee \sigma_{age < 26}$? 50% +46%-(50% x 46%) = 73%
- Database Design: requirements analysis, conceptual design, logical design, schema refinement, physical design, security design
- Data independence? **Hellerstein's Inequality**: If your env changes much faster than application, you need data independence.

Data Models

- Entity set (rectangles), relationship set (diamond), attribute (oval)
- **Key constraints:** arrow, each dept has at most 1 manager
- **Participation constraints:** At least 1. Every employee works in a dept
- **Weak entity** can be identified uniquely by id of another entity





- **x->y** means x determines y
- **Superkey** a set of columns in a table that determines all.
- **Candidate key:** a minimal set of columns that determine all.
- **Primary key:** a single chosen candidate key
- **Update anomaly:** can't update just 1 column. **Insertion anomaly:** insert, but don't know wage, **deletion anomaly,** delete all employees with certain hour, we lose the info about wage