Merge-Join

First we will consider equijoins on primary index keys.

```
create table customers (cid … primary key
                        …    )
create table orders (oid    …,
                     cid … primary key,
                        …)
```


Merge-Join

equijoins on primary index keys.

```
create table customers (cid … primary key
                        …    )
create table orders (oid    …,
                     cid … primary key,
                        …)
```

- Data is sorted & clustered on disk… by primary key
  - So how does it come off of the disk?
    - HDDs
    - SSDs
  - I call this "**The Golden Case**" for merge-join

Merge Join

Thanks:
https://sqlity.net/en/1480/a-join-a-day-the-sort-merge-join/

Ann	1
Ben	2
Cal	4
Dan	5
Emi	7
Fin	9

1	8377
4	9476
6	2735
7	1884
8	9546
9	4572

Careful:

- Above works for join arguments being unique.
  //no duplicate values in the join argument
- Modify algorithm to allow for duplicate values
  // left as an exercise

Merge-Join Pseudo Code

Assume block structure is buried in iterator abstraction

```
Merge-Join(relation R, relation S, attr rjk, attr sjk ){
   r = open(R, 1)   // read a block of R
   s = open(S, 1)   // read a block of S
   until( r = EOF or s = EOF) {
         if (r.rjk = s.rjk) {output(r,s), r = next(r)}
         if (r.rjk < s.rjk) {r = next(r)}
         if (r.rjk > s.rjk) {s = next(s)}
   }}
```


Cost of Merge-Join

B(R) + B(S) // per text

What happens in real life?

- Hint: consider the affine cost model

What if an equijoin, but not on index key?

Is there a secondary index on the join argument?

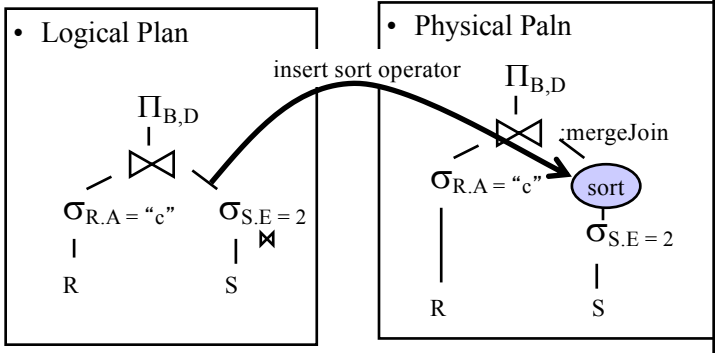
Do I want to use the secondary index?

Assume No Index

Recall two phase sorting

- for all practical purposes relations can be sorted in, 2 reads + 1 write
  - (final write doesn't count, by convention)
- cost to sort R = 3 B(R) // why 3 and not 4?

Optimizer will Transform the Plan



Merge-Sort Cost – No Index

5(B(R) + B(S))

- 3x to sort
- 1x (did have to write the sorted results out)
- 1x for merge join

May feel high – but it is very predictable

Many Opportunities to be Clever

- what if 1 relation fits in memory?
  - sort is 1*B(R)
- integrate merge-join with second phase of two phase sort,
  - 3(B(R) + B(S))