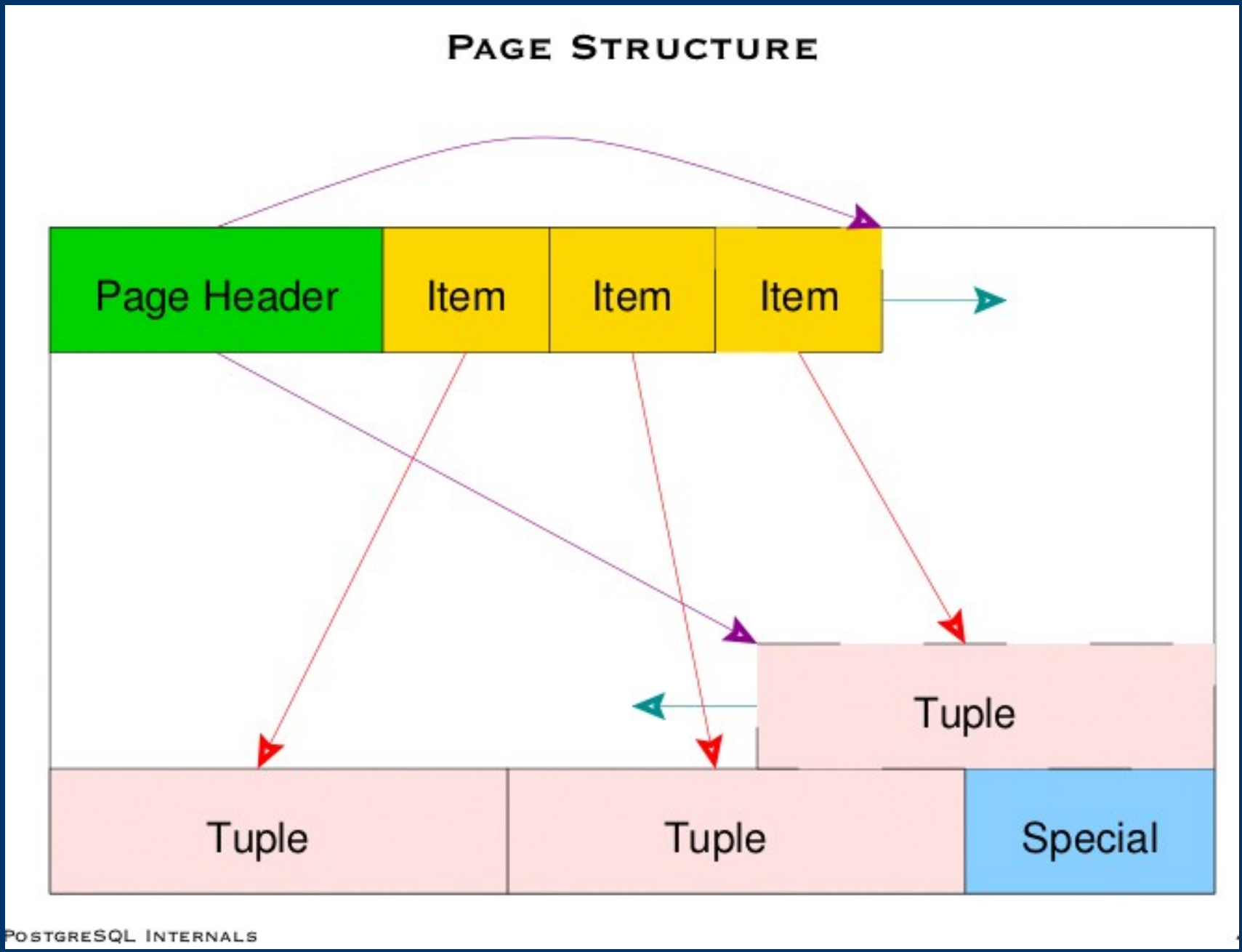
How is data stored?

- Table and index Data are stored in blocks(aka Page).
- All IO is done at least one block at a time.
- Typical block size is 8Kb.
- Each table contains several rows/tuples.(Deliberately ignoring very large rows).

Points of language

Oracle	Postgresql
Block	Page
Row	Tuple
Full Table Scan	Sequential Scan

The original plural form of index was indicies.
however most people today use indexes
and it is has been accepted into the English language,
I will use indexes.

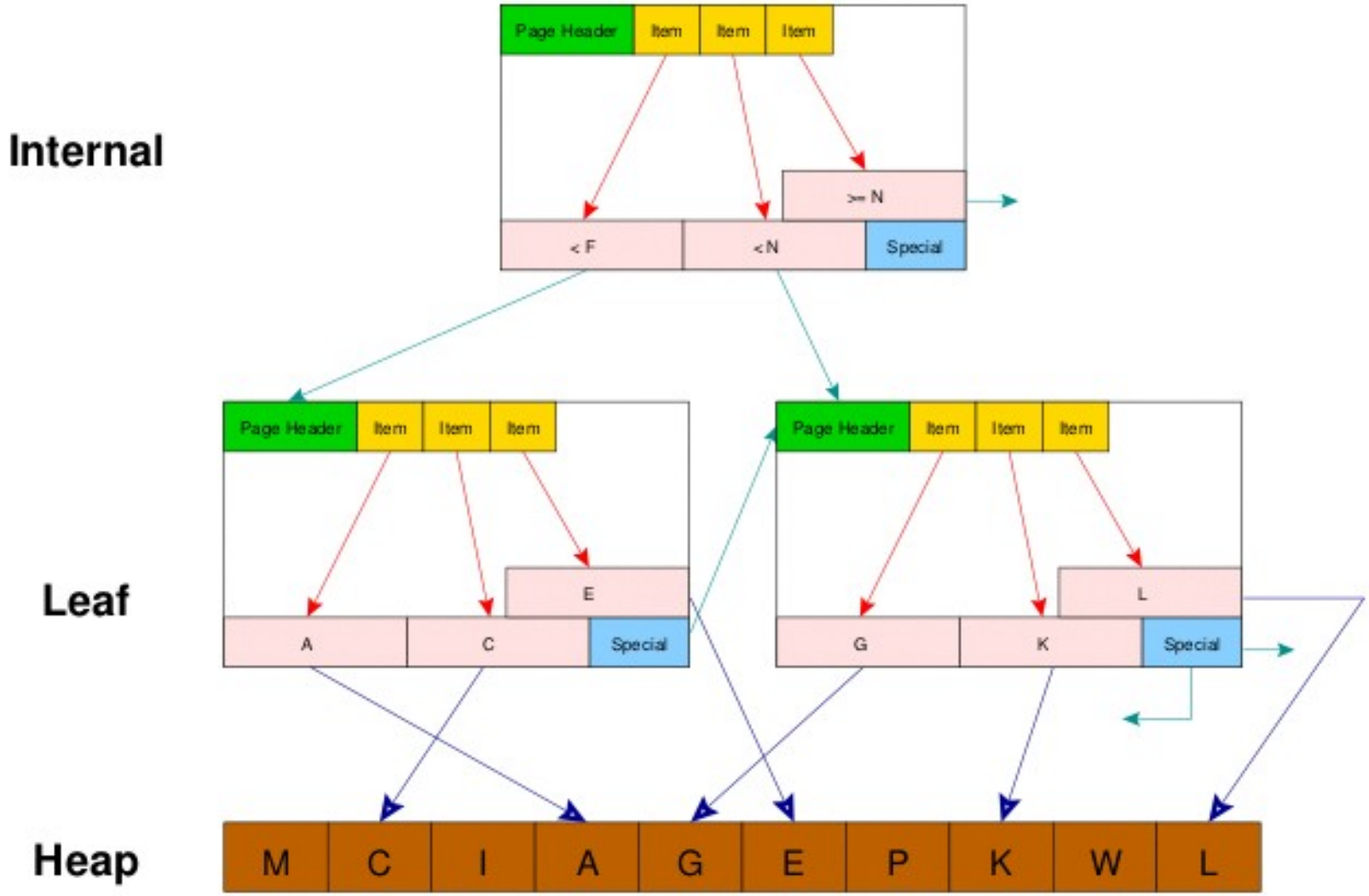


Oracle Has the the Header at the End

What is an Index

- An Index contains an organized summary of information in a table together with references to the location of rows in the table.(a rowid).
- The most common form for an index is the balanced tree Index, from here on referred to as B-Tree index.
- There are other types of indexes, mainly Bitmap indexes.

Index Page Structure



B-Tree Indexes

- Data(references to rows) is only in the leaves.
- Very Large Branching factor. each Node may Split into hundreds of nodes at the next level.
- Most indexes have a depth of no more than 3.
- Not truly Balanced, in order to avoid expensive on line balancing operations, The tree may slowly grow unbalanced or even increase in size unproportionally to table size.
- Each index indexes only a few table columns.

Access Path: Scan Methods

- Sequential Scan(aka Full Table Scan, FTS)
- Index Scan(in Oracle Index Range/unique scan)
- Index Fast Full Scan – Oracle Only.

Sequential Scan

- Reads All data in table. Regardless of number of rows actually retrieved.
- Takes full advantage of reading sequencially from Magnetic Hard drives.
- May Read Empty blocks.
- Unordered output.

Index Scan

- Traverse the index first, accordingly to part of the where clause. fetch from table only relevant rows.
- Output rows are ordered according to the index.
- When fetching many rows is likely to perform many random access reads from disk.

Index Only access

- Oracle(but not postgres) allows queries to run only on the index with out retrieving data from the Heap.
- Works only when querying only indexed columns.
- Postgresql requires reading the Heap any way to know tuple visibility.
- This allows Oracle to perform Index Fast Full Scan(index_ffs).

How much does it cost?

- Full Table Scan:
  - startup cost: 0
  - total: number of blocks in table.
- Index Scan:
  - startup cost:0
  - total: tree hight + number of leaf blocks read + number of Heap blocks read.

Full Table Scan Cost

- Since the table blocks are stored more or less sequentially on disk we can read them quickly.
- We will set a DB parameter for how many blocks should we read at once when performing sequential reads, DB_BLOCK_MULTIREAD_COUNT.
- We will shorten it here to <Mutli>
- Full Table Scan will cost us:
  - <BLOCKS>/<Multi>

Index Scan Cost

- let the Filter Factor, <FF>, be the percentage of the records from tree read in the tree scan
- let the Screening Factor,<SF>, be the percentage of the heap records read.
- let <LEAFS> be the number of leafs in the index.
- let <BLOCKS> be the number of blocks in a table.
- the total cost of an index scan is:
  - <HEIGHT>-1+<FF>*<LEAFS>+<SF>*<BLOCKS>

The Clustering Factor

- Our previous formula is inaccurate, since when scanning the tree, the heap is not referenced sequentially and we may end up reading a block for each row we fetch. (if the data in the table is totally unordered with respect to the index).
- The previous formula is correct only if the data in the table happens to be ordered in the same order as the index.

The Clustering Factor. cont.

- We will designate the number of block changes when scanning the entire index as the clustering factor.
- The clustering factor ranges from the number of blocks in the table, to the number of rows in the table.
- We will designate the Clustering Factor: <CF>
- A revised formula will be:
  - <HEIGHT>-1+<FF>*<LEAFS>+<SF>*<CF>

Clustering Factor and DB cache

- However since the Database has a significant cache it is likely that when scanning an index and reading the same block twice it would be in the cache.
- Let OPTIMIZER_INDEX_COST_ADJUST be the percentage of heap blocks **not** found in cache when scanning an index. for short: <adjust>.
- Our Final formula would be:
  - <HEIGHT>-1+<FF>*<LEAFS>+<SF>*<CF>*<adjust>

Join Methods

- When querying data from more than one table a join is used:
- There Are several methods of joining tables:
  - Nested loop Join
  - Sort Merge Join
  - Hash Join

Nested Loops

- Iterate over relevant rows from primary table, for each row fetch matching row(s) from secondary table.
- Cost is the Cost to access the primary table + number of rows fetched from primary table * cost of fetching the matching rows from secondary table.
- Sensitive to ordering of the tables.

Sort Merge Join

- Scan both tables ordered by the join columns, matching the rows as you go.
- Cost is the Time to read(and sort) both tables.
- If the table may be accessed through an index ordered by the join column sorting is unnecessary.

Hash Join

- Load the Secondary table into a hash table, with the join column as the key.
- Scan the primary table and fetch matching rows from from the hash table.
- works only for equijoin.(using equals sign).
- Cost is time to read both tables(when secondary resides can reside in RAM).

The CBO

- The CBO needs to find optimal path for a given query.
- If there are a small number of tables (less than ~6) it will estimate the cost of **all** join&access methods.
- If there are many tables joined a genetic heuristic algorithm is used.
- An access path is a tree of join methods and accees methods as well as extra operations such as filter, sort and aggregate.

The CBO , Continued

- For every stage the CBO estimates:
  - the cost for startup.
  - the cost for fetching all rows.
  - number of records.
- The CBO makes these estimates based on statistics it gathers on the Tables and indexes used.

Lies, Damn Lies and Statistics

- Table Statistics normally include:
  - number of blocks.
  - number of rows
  - for each column, number of distinct values
  - for each column, minimum&maximum(sometimes more of a histogram is available)
- Index statistics normally include:
  - index height.
  - number of leaf blocks.
  - Clustering Factor. (to what extent are the table and index ordered the same).

Estimating Filter&Screening factor

- for equals clause assume FF=1/<num distinct>
  - or use full histogram if available.
- for range clause assume linear dispersion withing histogram buckets.
- If query has bind variables the values are not known during parse time, in which case magic numbers are used for range clauses(9% for '>', 4% from between or like).

Optimizer pitfalls

- The optimizer does not know how columns are correlated.
- When using pre parsed query, the optimizer does not have the values for bind variables.
- When the Values are from a joined table the optimizer does not have real values.
- Histograms are normally very incomplete even when usable
- Statistics may be out of date(or missing).

Solutions

- Rewrite query.
  - total rewrite
  - change join to subquery and vice-versa.
  - add redundant clauses.
- use optimizer hints.
- Dynamic query replanning – work in progress(IBM).

Bitmap indexes

- A bitmap index contains a bitmap of the table rows for each value the indexed columns have.
- If we have a Bitmap index on the Gendre column 2 bitmaps will be created each containing a 1 for the rows that have that value.
- Useful for DataWhareHouse applications.
- Expensive to maintain.(insert/update/delete underlying table).
- several bitmap indexes can be used together in one query.