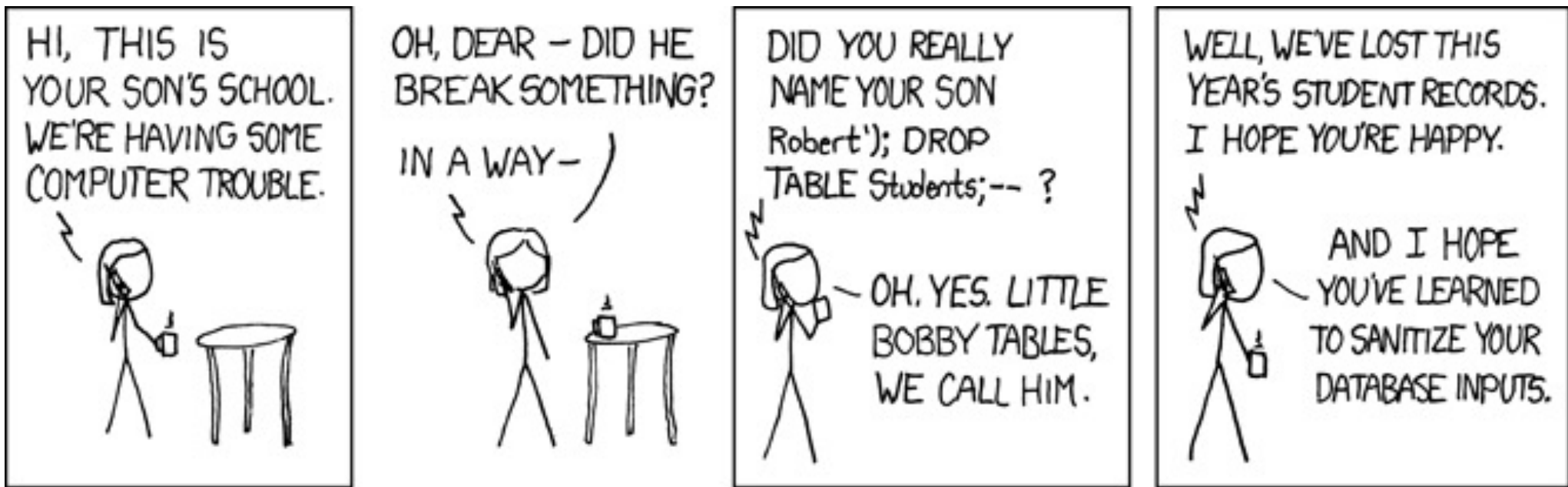


Constraints, Indices, B-Trees

Maintaining Integrity of Data

- You are creating a search engine for Rhodes' website, called Rhoogle.
- You have an SQL query:
 - "SELECT * FROM pages WHERE name=" + VAR + "";"



Maintaining Integrity of Data

- Data is *dirty*.
- How does an application ensure that a database modification does not corrupt the tables?
- Two approaches:
 - Application programs check that database modifications are consistent.
 - Use the features provided by SQL.

Integrity Checking in SQL

- Data type constraints (including NOT NULL).
- PRIMARY KEY and UNIQUE constraints.
- FOREIGN KEY constraints.
- Constraints on attributes and tuples.
- Triggers (schema-level constraints).

Constraints and Queries

- Often, constraints involve attributes we often perform searches (SQL SELECTs) on.
- To speed up queries, DBs will often create *indices* automatically for you.

Indexes

- *Index* = data structure used to speed access to tuples of a relation, given values of one or more attributes.

Declaring Indexes

- No standard!
- Typical syntax:

```
CREATE INDEX MovieIdx ON  
  Movie (MovieId);
```

```
CREATE INDEX CastsIdx ON  
  Casts (ActorId, MovieId);
```

Types of Indexes

- **Primary:** index on a key
 - Used to enforce constraints
- **Secondary:** index on non-key attribute

Using Indexes: Equality Searches

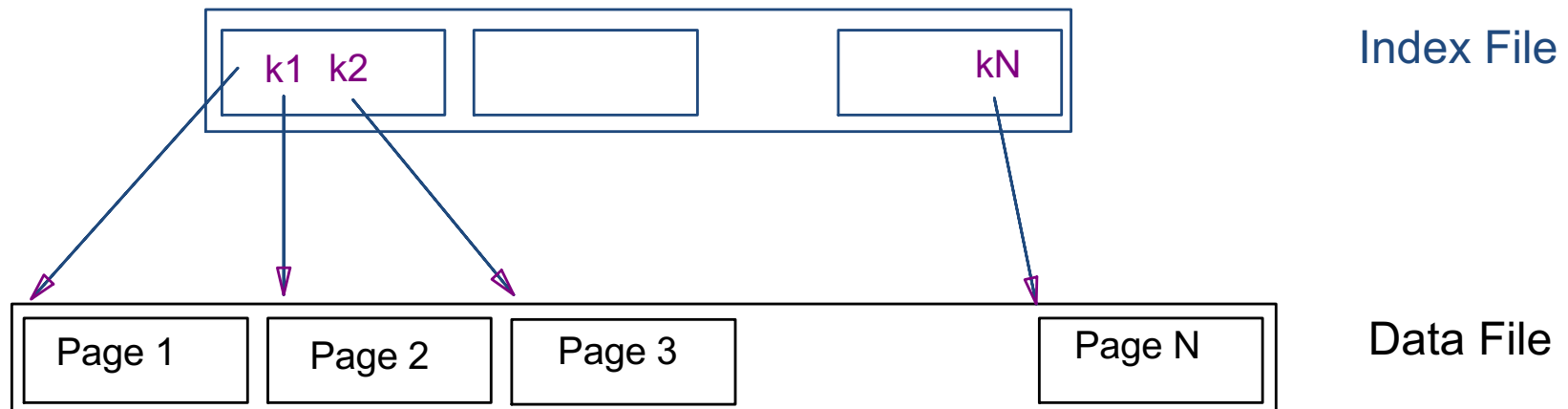
- Given a value v , the index takes us to only those tuples that have v in the attribute(s) of the index.
- What data structure would be useful here?

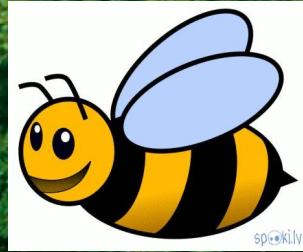
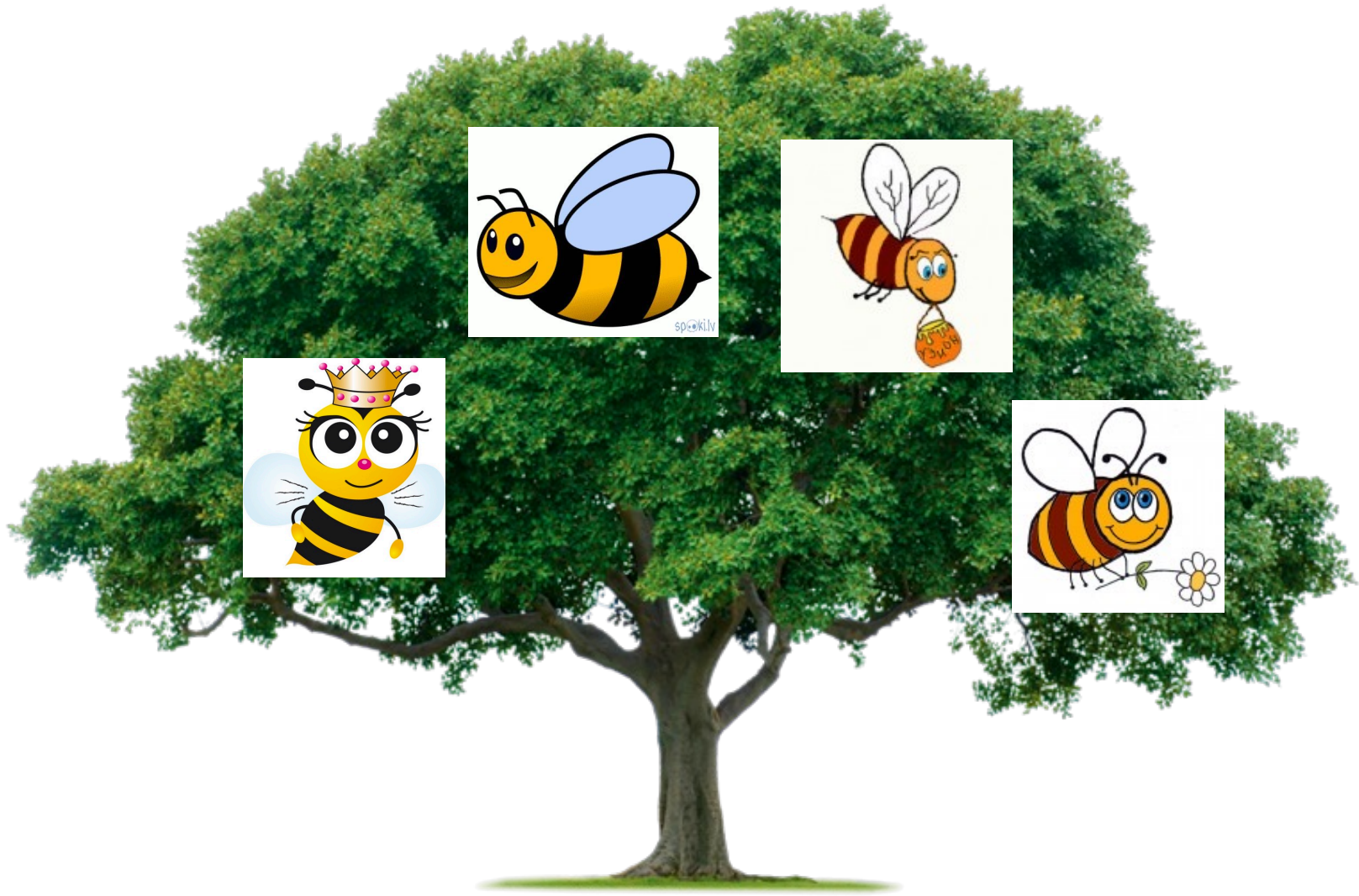
Using Indexes: Range Searches

- "Find all students with GPA > 3.0"
- What data structure(s) work here?

Range Searches

- *"Find all students with GPA > 3.0"*
- May be slow, even on sorted file
- Solution: Create an index file.





B-trees

- Extension of binary search trees to n-way search trees (where $n > 2$)
- Balanced (like red-black trees)

Why B-Trees Are So Great for DB Indexes

- DBs are usually on disk, not RAM
 - B-tree structure aligns with *disk pages*
 - Hierarchical structure minimizes number of disk reads.
- Keeps info in sorted order for equality or range searches.
- Balanced tree structure gives fast searches, insertions, deletions.

Definition

- B-tree of **order d** is a tree with these properties:
 - Internal nodes have one more child (pointer) than data elements (keys). Leaf nodes have no children.
 - Root has between 1 and $2d$ data elements.
 - Non-root nodes have between d and $2d$ elements.
 - All leaves are at the same depth in the tree.
 - Has ***extended search property*** (binary search tree property extended to multiway tree)

Algorithms: Search

- Extrapolated from binary tree search algorithm.

Algorithms: Insert

- First, find *leaf* node where data would go.
- Insert(data, node):
 - If data can fit in node, add it to the node.
 - If causes overflow:
 - split node at the median value.
 - Everything less than median becomes new leaf node.
 - Everything greater than median becomes new leaf node.
 - Promote median to parent node; call insert(median, parent) [*may create new parent node if there is no parent*]

Algorithms: Delete

- Search for item to delete
- If at leaf node, delete the item
 - Rebalance up from leaf if necessary
- If at internal node, swap with largest child in left sub-tree (analogous to BST deletion swap)
 - Rebalance if necessary