

Describe the concepts of declarative and procedural query languages.

In a declarative query language, the user specifies what information is needed without detailing how to obtain it. It focuses on the "what" rather than the "how."

**Example:** SQL (Structured Query Language) is a prime example of a declarative query language. In SQL, you declare the desired outcome by writing a query, and the database management system figures out the most efficient way to execute it. For instance, consider the following SQL query:

SELECT FirstName, LastName FROM Employees WHERE Department = 'IT';

**Advantages:**

* Users can focus on the logic of the query without concerning themselves with the underlying implementation details.
* It allows for optimization by the database management system, which can choose the most efficient execution plan.

**Procedural Query Language:**

* **Definition:** In a procedural query language, the user specifies not only what information is needed but also how to obtain it. It involves a step-by-step procedure to achieve the desired outcome.

**Example:** The programming language PL/pgSQL (Procedural Language/PostgreSQL) is an example of a procedural query language. In PL/pgSQL, you can define a series of steps or procedures to accomplish a specific task. Here's a simplified example:

DECLARE

employee\_record RECORD;

BEGIN

FOR employee\_record IN (SELECT \* FROM Employees WHERE Department = 'IT') LOOP

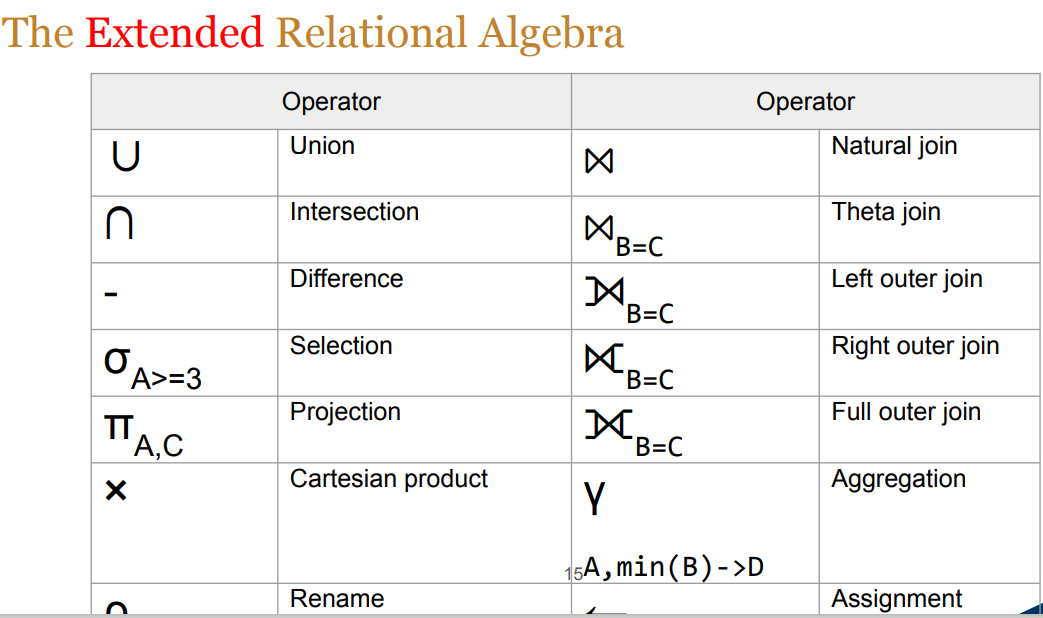
-- Process each employee record

RAISE NOTICE 'Employee: % %', employee\_record.FirstName, employee\_record.LastName;

END LOOP;

END;

Describe the different relational algebra (and extended RA) operators.

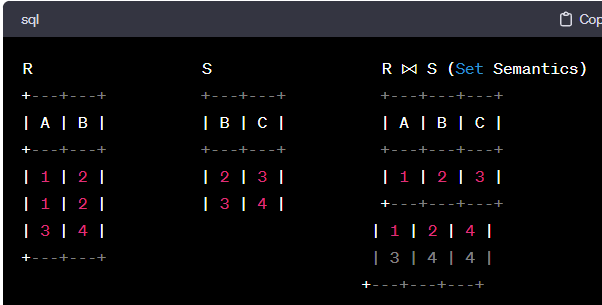


Differentiate the set and bag semantics in RA, and describe how the query results may differ applying one semantic or the other.

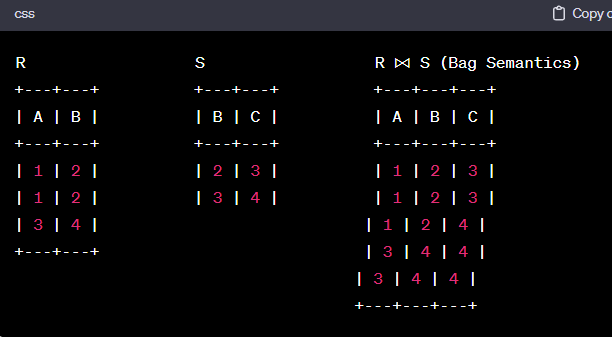
In the context of Relational Algebra (RA), set semantics and bag semantics refer to two different approaches in handling duplicate values in the results of relational operations.

In the context of Relational Algebra (RA), set semantics and bag semantics refer to two different approaches in handling duplicate values in the results of relational operations.

1. **Set Semantics:**
   * **Definition:** Set semantics treat relations as sets, meaning that each tuple within a relation is unique. In other words, duplicate tuples are not allowed in the result set.
   * **Example:** Consider the natural join operation in set semantics. If there are duplicate tuples in the join result, they will be eliminated, and only unique tuples will be retained in the output.| 3 | 4 | 4 | +---+---+---+
   * **Characteristics:**
     + Result sets do not contain duplicate tuples.
     + Operations like union, intersection, and difference are based on set theory.



1. **Bag Semantics:**
   * **Definition:** Bag semantics, on the other hand, allow duplicate tuples in the result set. A bag is also known as a multiset, and it can contain multiple occurrences of the same tuple.
   * **Example:** Using the same example as before, if bag semantics are applied to the natural join operation, duplicate tuples are retained in the output.



Illustrate the transformation of a sub-query into a join then into an RA expression.

SELECT \*

FROM Orders

WHERE CustomerID IN (SELECT CustomerID FROM Customers WHERE CustomerName = 'John Doe');

SELECT Orders.\*

FROM Orders

JOIN Customers ON Orders.CustomerID = Customers.CustomerID

WHERE Customers.CustomerName = 'John Doe';

Describe the role of statistics in cost-based query optimization

1. **Cardinality Estimation:**
   * Statistics provide information about the distribution of data in the tables, including the number of distinct values in each column. This helps the query optimizer estimate the cardinality, or the number of rows returned by a particular operation. Accurate cardinality estimates are crucial for choosing the most efficient join algorithms, access methods, and other operations in the query plan.
2. **Selectivity Estimation:**
   * Selectivity refers to the fraction of rows in a table that satisfy a given predicate. Statistics help estimate the selectivity of different predicates in the WHERE clause, guiding the optimizer in choosing the most selective and efficient access paths. This information is vital for deciding whether to use an index or perform a full table scan.

**Index Selection:**

* Statistics on the distribution of data in indexed columns help the optimizer decide when to use an index and which index to use. The optimizer considers factors like the selectivity of the index, the distribution of values, and the overall cost of using the index versus scanning the entire table.

**Join Order and Join Methods:**

* Statistics assist in determining the order in which tables are joined in a query. By estimating the cardinality of intermediate results, the optimizer can evaluate different join orders and choose the one with the lowest expected cost. Statistics also play a role in selecting the appropriate join methods (e.g., nested loop join, hash join) based on the size of the tables being joined.

Discuss the use of histogram for attribute statistics

Helps to understand distribution of column values

Explain the use of indexes in query processing

Indexes in query processing improve the speed of data retrieval by providing a quick and efficient way to locate specific rows in a database table based on the values in one or more columns.

1. Explain the concepts of sequential file, dense index, sparse index, 1st level index, and 2nd level index, secondary index

**Sequential File:**

* + A sequential file is a type of data storage where records are stored in a consecutive order based on a primary key or some predetermined order. Accessing data in a sequential file involves reading records one after the other in the order they are stored.

1. **Dense Index:**
   * A dense index is an index structure that contains an entry for every record in the data file, indicating the location of each record. It provides a direct mapping between the index entries and the corresponding records, making lookups efficient.
2. **Sparse Index:**
   * A sparse index is an index structure where not every record in the data file has a corresponding entry in the index. Instead, the index includes entries for only a subset of the records, typically at regular intervals. This reduces the size of the index but requires additional effort for locating specific records.

Secondary Index

A primary index sorts the data file by its search key. The search key DOES NOT have to be the same as the primary key. ⇨A secondary index does not determine the organization of the data file. ⇨A single-level index has only one index level.

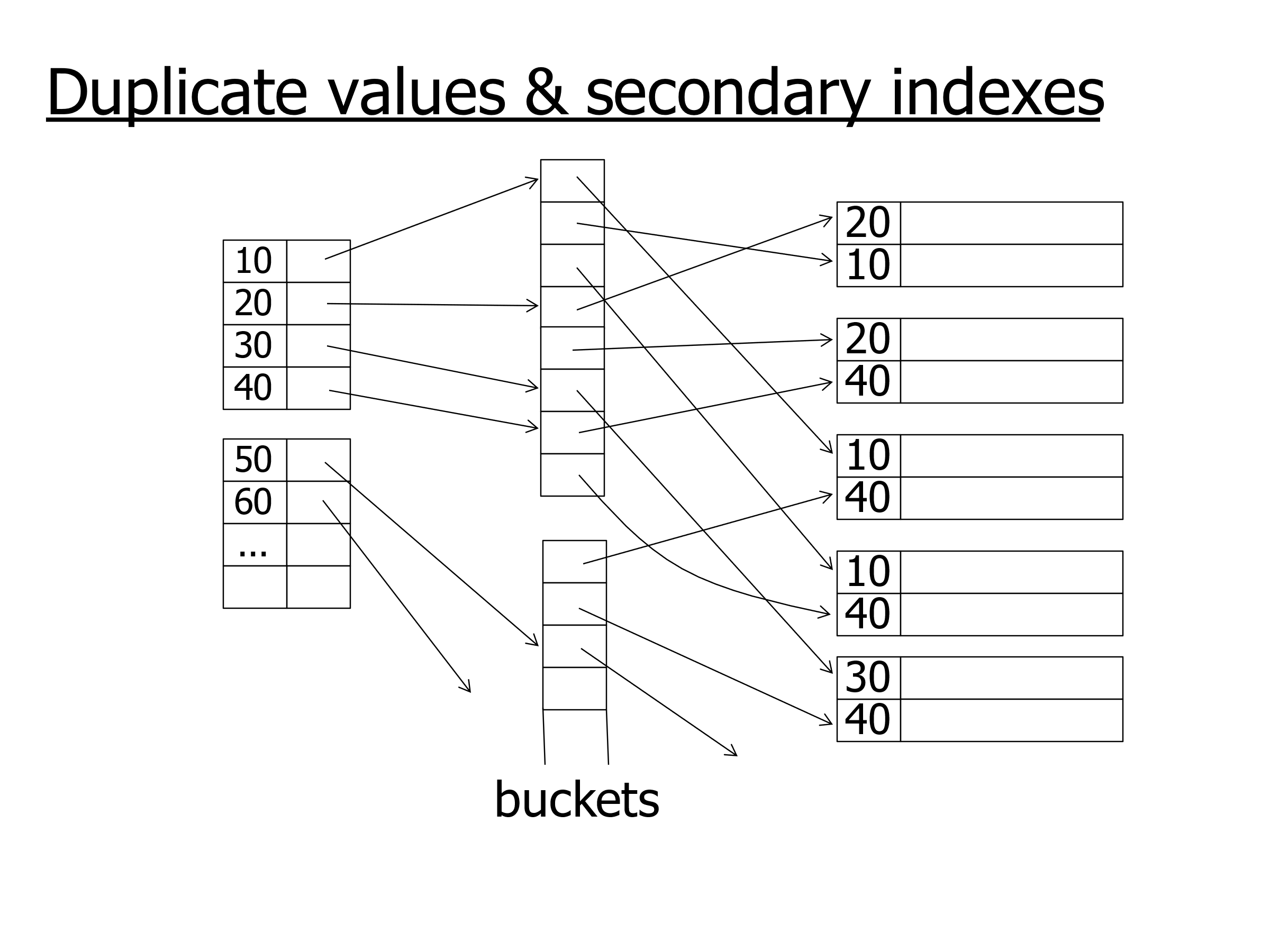
Indexes other than the clustered index are known as secondary indexes.

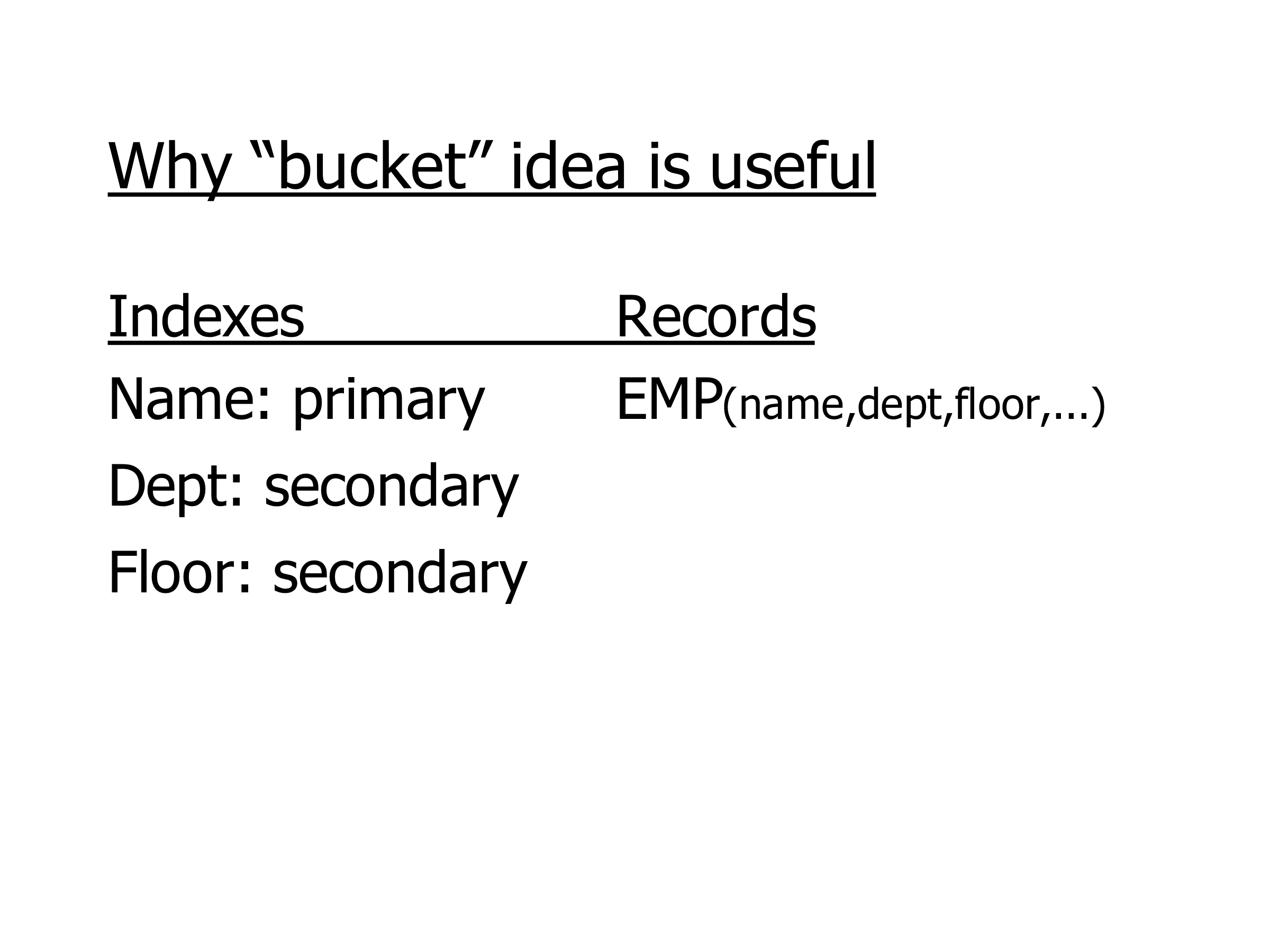
Multilevel index : A single index for a large size data file increases the size of index table and increases the search time that results in slower searches. The idea behind multilevel indexes is that, a single level index is divided into multiple levels, which reduces search time. In multilevel indexes, the first level index consists of two fields, the first field consists of a value of search key attributes and a second field consists of a pointer to the block (or second level index) which consists that value and so on. To search a record in multilevel index, binary search is used to find the largest of all the small value or equal to the one that needs to be searched. The pointer points to a block of the inner index. After reaching to the desired block, the desired record is searched (in case of two-level indexing) otherwise again the largest of the small values or equal to the one that needs to be searched and so no.

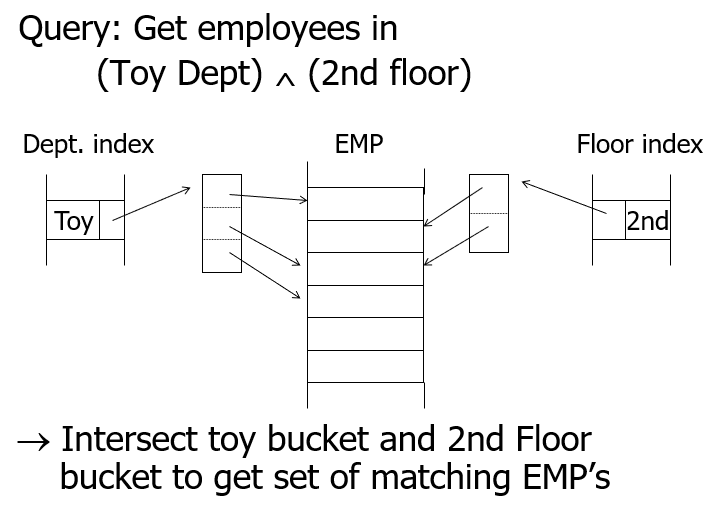
Duplicate entries

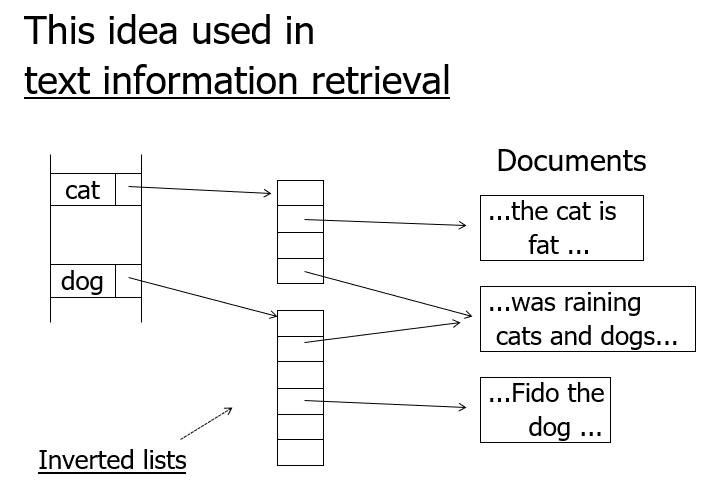
Dense index with one entry with key K for each record of the data file that has search key K. That is, we allow duplicate search keys in the index file. Finding all the records with a given search key K is thus simple: Look for the first K in the index file, find all the other K's, which must immediately follow, and pursue all the associated pointers to find the records with search key K

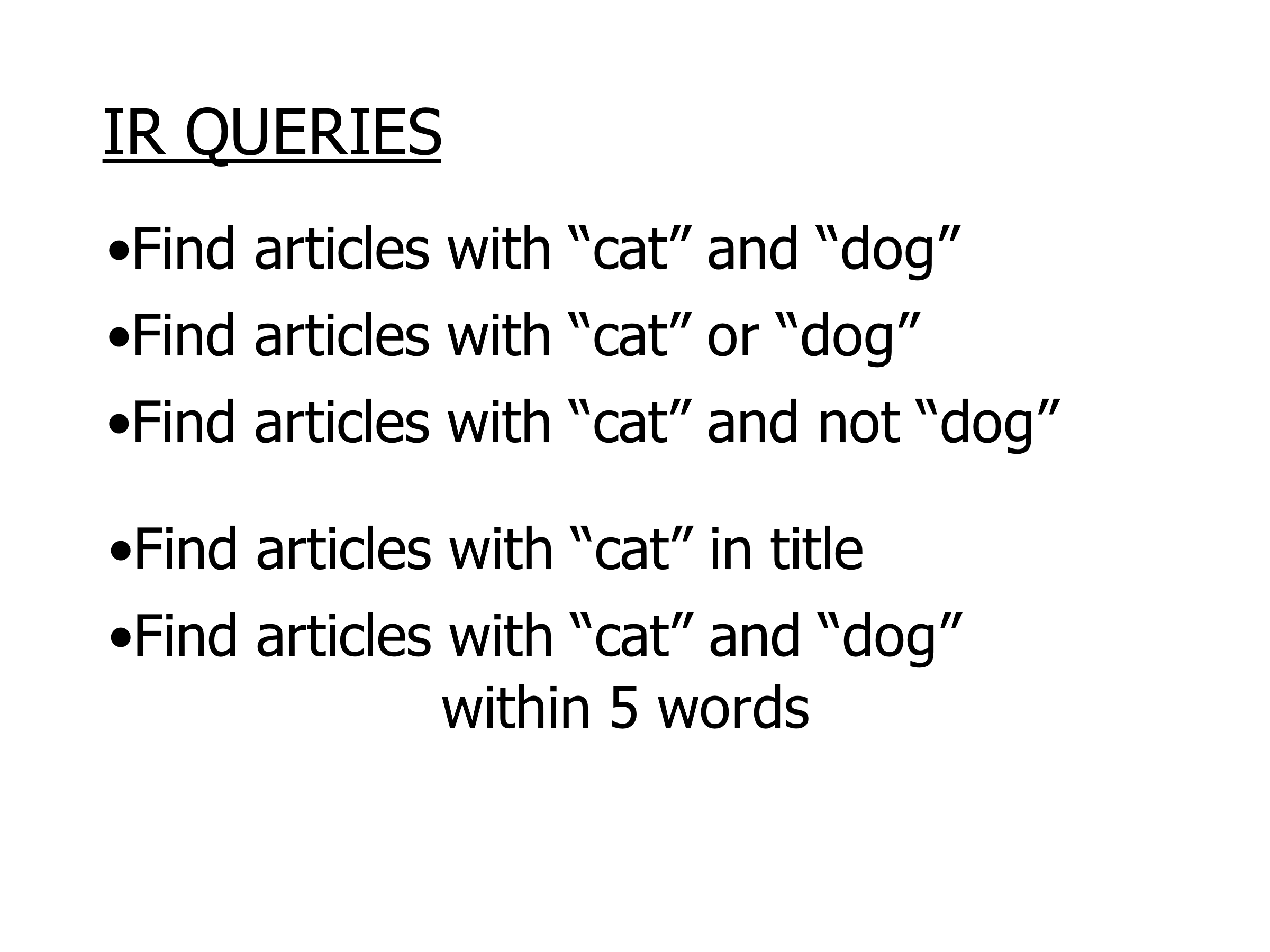
Illustrate the benefits of buckets in secondary indexes.











Illustrate the properties of the Btree that allows us to answer inequality (<=, >, >=) or range searches (between) efficiently.

The two important properties are: (1) the keys stored in a Btree are stored and (2) the nodes contain pointers to the previous and next node in the same level of the tree. We can then for example answer an inequality search by first looking for the boundary value using the equality condition and then iterate along the leaf nodes in the correct direction to return all the searched rows.

Illustrate the insertion/deletion strategies in conventional indexes, also for the case of duplicate keys