Index-Organized Tables

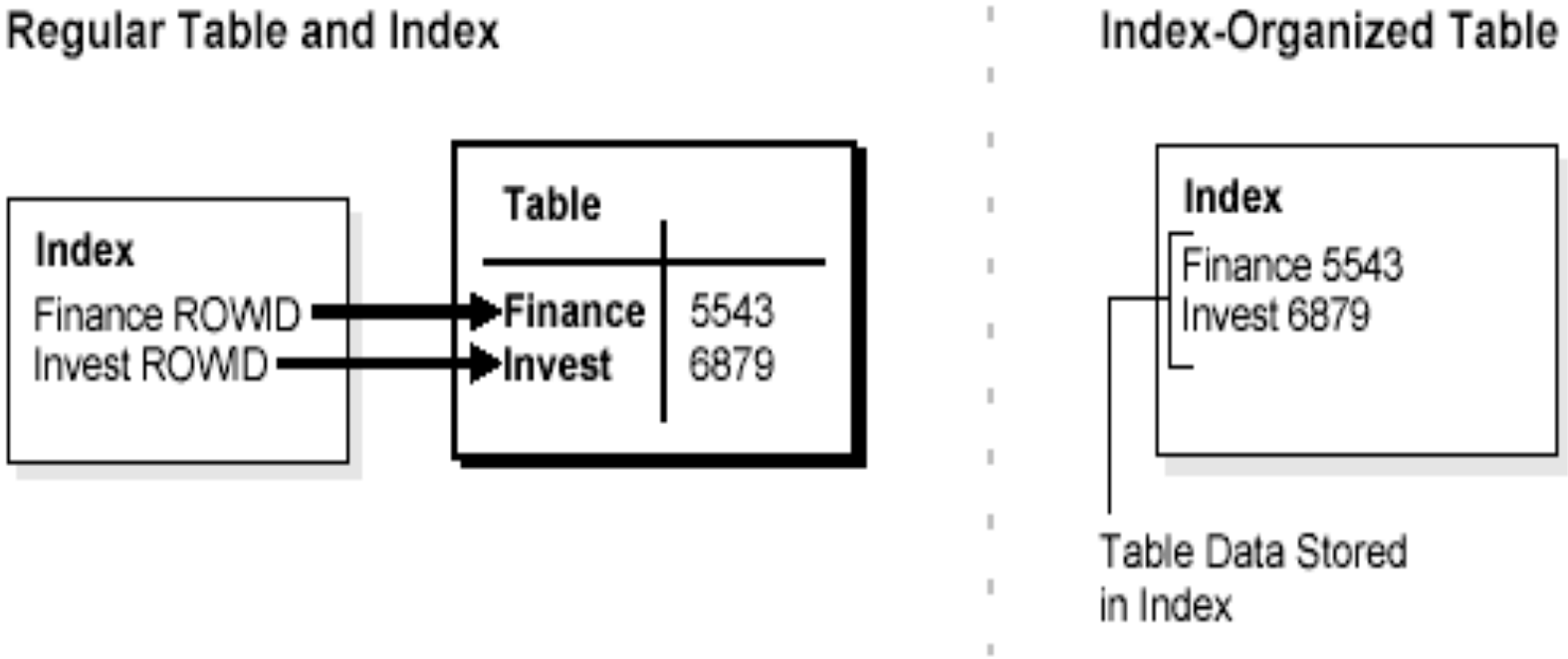
Introduction

- An **index-organized table** has a storage organization that is a variant of a primary B-tree.
- Besides storing the primary key column values of an index-organized table row, each index entry in the B-tree stores the non-key column values as well.
- Data for an index-organized table is stored in a B-tree index structure in a primary key sorted manner.
- Rather than having a row's rowid stored in the index entry, the non-key column values are stored.

  Thus, each B-tree index entry contains

  `<primary_key_value, non_primary_key_column_values>.`

Introduction



- Applications manipulate the index-organized table just like an ordinary table, using SQL statements. However, the database system performs all operations by manipulating the corresponding B-tree index.

Benefits of Index-Organized Tables

- Index-organized tables provide ***faster access*** to table rows by the primary key or any key that is a valid prefix of the primary key.
- Presence of non-key columns of a row in the B-tree leaf block itself ***avoids an additional block access***.
- In order to allow even faster access to frequently accessed columns, you can use a row *overflow storage* option to push out infrequently accessed non-key columns from the B-tree leaf block to an optional overflow storage area. This allows limiting the size and content of the portion of a row that is actually stored in the B-tree leaf block, which may lead to a higher number of rows in each leaf block and a ***smaller B-tree***.
- Unlike a configuration of ordinary table with a primary key index where primary key columns are stored both in the table and in the index, there is ***no such duplication*** here because primary key column values are stored only in the B-tree index.

Index-Organized Tables with Row Overflow Area

- In index-organized tables, the B-tree index entries can be large, because they consist of the entire row. This may destroy the dense clustering property of the B-tree index.
- Oracle provides the `OVERFLOW` clause to handle this problem. We can specify an overflow tablespace so that, if necessary, a row can be divided into the following two parts that are then stored in the index and in the overflow storage area, respectively:
  - The index entry, containing column values for all the primary key columns, a physical rowid that points to the overflow part of the row, and optionally a few of the non-key columns,
  - The overflow part, containing column values for the remaining non-key columns

Index-Organized Tables with Row Overflow Area

- With `OVERFLOW,` you can use two clauses, `PCTTHRESHOLD` and `INCLUDING,` to control how Oracle determines whether a row should be stored in two parts and if so, at which non-key column to break the row.
- Using `PCTTHRESHOLD,` you can specify a threshold value as a percentage of the block size. If all the non-key column values can be accommodated within the specified size limit, the row will not be broken into two parts. Otherwise, starting with the first non-key column that cannot be accommodated, the rest of the non-key columns are all stored in the row overflow storage area for the table.
- The `INCLUDING` clause lets you specify a column name so that any non-key column, appearing in the `CREATE TABLE` statement after that specified column, will be stored in the row overflow storage area.

Secondary Indexes on Index-Organized Tables

- Secondary index support on index-organized tables provides efficient access to index-organized table using columns that are not the primary key nor a prefix of the primary key.
- Oracle constructs secondary indexes on index-organized tables using logical row identifiers (*logical rowids*) that are based on the table's primary key.

Bitmap Indexes on Index-Organized Tables

- Oracle9*i*, Release 1 (9.0.1), supports bitmap indexes on index-organized tables.
- A mapping table is required for creating bitmap indexes on an index-organized table.
- The mapping table provides one-to-one mapping between logical rowids of the index-organized table rows and physical rowids of the mapping table rows.
- In both heap-organized and index-organized base tables, a bitmap index is accessed using a search key. If the key is found, the bitmap entry is converted to a physical rowid. In the case of heap-organized table, this physical rowid is then used to access the base table. However, in the case of index-organized table, the physical rowid is then used to access the mapping table. The access to the mapping table yields a logical rowid. This logical rowid is used to access the index-organized table.

Index-Organized Table Applications

- Online Transaction Processing (OLTP)
- Internet (for example, search engines and portals)
- E-Commerce (for example, electronic stores and catalogs)
- Data Warehousing
- Time-series applications