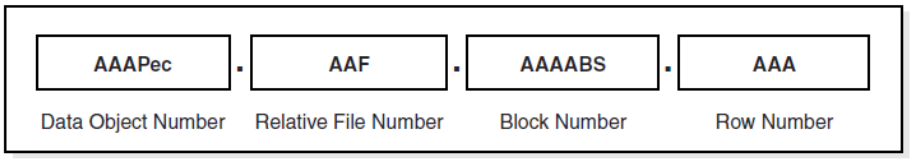
Oracle database concepts

ROWID

Oracle Database uses a **rowid** to uniquely identify a row. Internally, the rowid is a structure that holds information that the database needs to access a row. A rowid is not physically stored in the database, but is inferred from the file and block on which the data is stored.

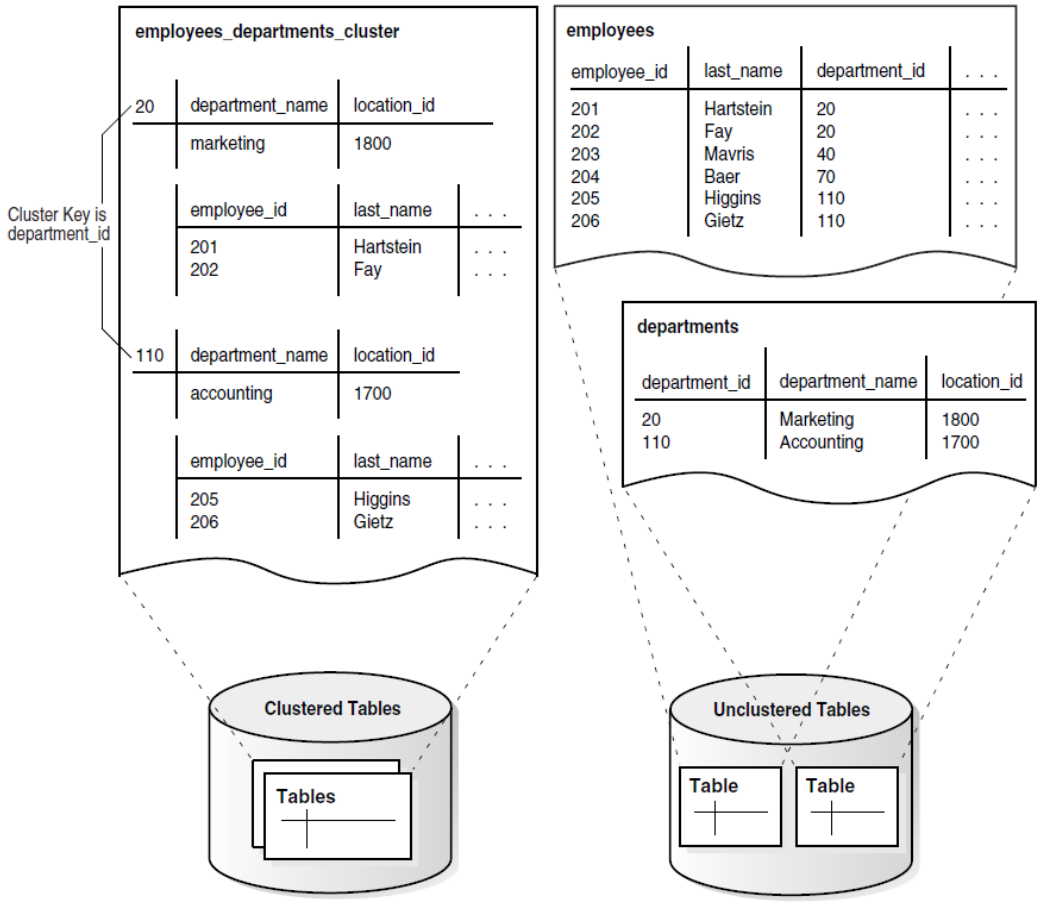
Figure 12–8 ROWID Format



A single data segment in a database stores the data for one user object. There are different types of segments. Examples of user segments include:
Table, table partition, table cluster, LOB or LOB partition**, Index** or **index partition**.

A **table cluster** is a group of tables that share common columns and store related data in the same blocks. When tables are clustered, a single data block can contain rows from multiple tables. For example, a block can store rows from both the employees and departments tables rather than from only a single table.
The cluster key is the column or columns that the clustered tables have in common. For example, the employees and departments tables share the department_id column. You specify the cluster key when creating the table cluster and when creating every table added to the table cluster.

Figure 2–6 Clustered Table Data

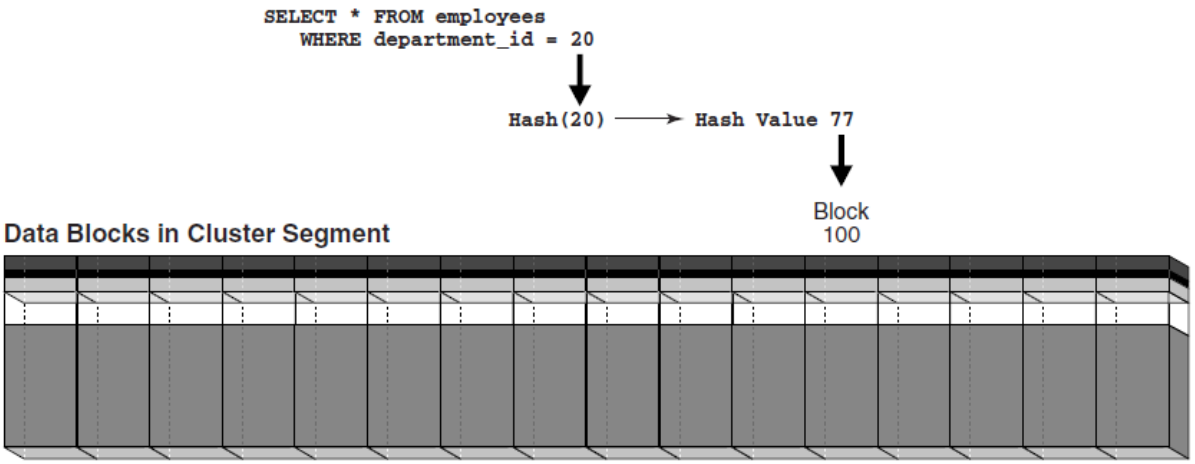


A **hash cluster** is like an indexed cluster, except the index key is replaced with a hash function. No separate cluster index exists. In a hash cluster, the data is the index.
With an indexed table or indexed cluster, Oracle Database locates table rows using key values stored in a separate index. To find or store a row in an indexed table or table cluster, the database must perform at least two I/Os:

- One or more I/Os to find or store the key value in the index
- Another I/O to read or write the row in the table or table cluster

To find or store a row in a hash cluster, Oracle Database applies the hash function to the cluster key value of the row. The resulting hash value corresponds to a data block in the cluster, which the database reads or writes on behalf of the issued statement.

Figure 2–7 Retrieving Data from a Hash Cluster



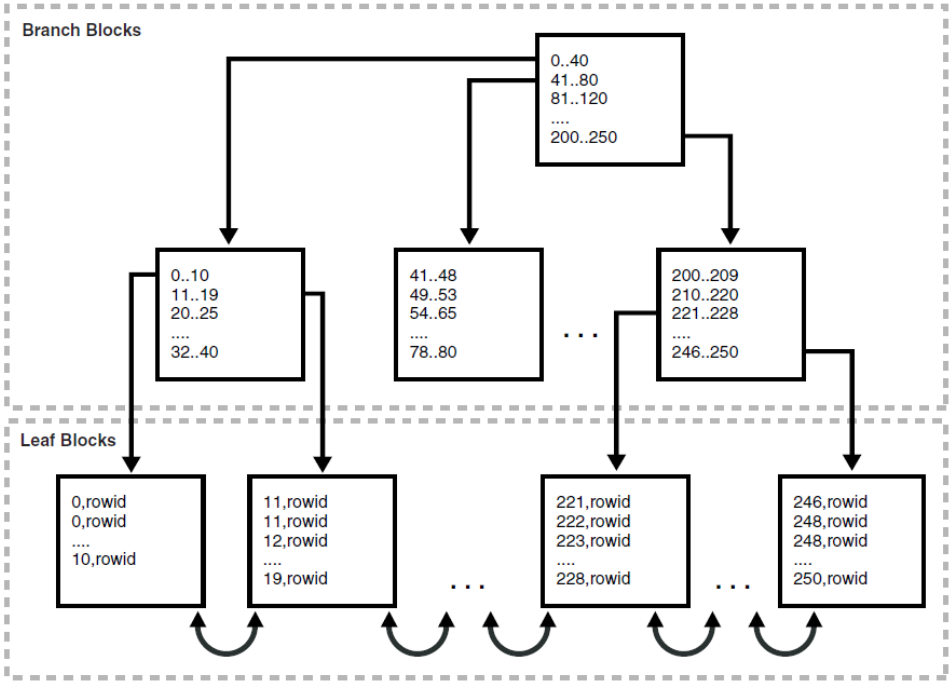
An **index** is an optional structure, associated with a table or table cluster, that can sometimes speed data access. By creating an index on one or more columns of a table, you gain the ability in some cases to retrieve a small set of randomly distributed rows from the table. Indexes are one of many means of reducing disk I/O.

A key is a set of columns or expressions on which you can build an index. Although the terms are often used interchangeably, indexes and keys are different. Indexes are structures stored in the database that users manage using SQL statements. Keys are strictly a logical concept.

A composite index, also called a concatenated index, is an index on multiple columns in a table.

A B-tree index has two types of blocks: branch blocks for searching and leaf blocks that store values. The upper-level branch blocks of a B-tree index contain index data that points to lower-level index blocks.

Figure 3–1 Internal Structure of a B-tree Index



Index Scan
In an index scan, the database retrieves a row by traversing the index, using the indexed column values specified by the statement. If the database scans the index for a value, then it will find this value in n I/Os where n is the height of the B-tree index. This is the basic principle behind Oracle Database indexes.

Full Index Scan
In a full index scan, the database reads the entire index in order.

Fast Full Index Scan
A fast full index scan is a full index scan in which the database accesses the data in the index itself without accessing the table, and the database reads the index blocks in no particular order.

Index Range Scan
An index range scan is an ordered scan of an index that has the following characteristics:
One or more leading columns of an index are specified in conditions.

Index Unique Scan
In contrast to an index range scan, an index unique scan must have either 0 or 1 rowid associated with an index key. The database performs a unique scan when a predicate references all of the columns in a UNIQUE index key using an equality operator. An index unique scan stops processing as soon as it finds the first record because no second record is possible.

Index Skip Scan
An index skip scan uses logical subindexes of a composite index. The database "skips" through a single index as if it were searching separate indexes. Skip scanning is beneficial if there are few distinct values in the leading column of a composite index and many distinct values in the nonleading key of the index.

A reverse key index is a type of B-tree index that physically reverses the bytes of each index key while keeping the column order. For example, if the index key is 20, and if the two bytes stored for this key in hexadecimal are C1,15 in a standard B-tree index, then a reverse key index stores the bytes as 15,C1. Reversing the key solves the problem of contention for leaf blocks in the right side of a B-tree index.

In a bitmap index, the database stores a bitmap for each index key. In a conventional B-tree index, one index entry points to a single row. In a bitmap index, each index key stores pointers to multiple rows.

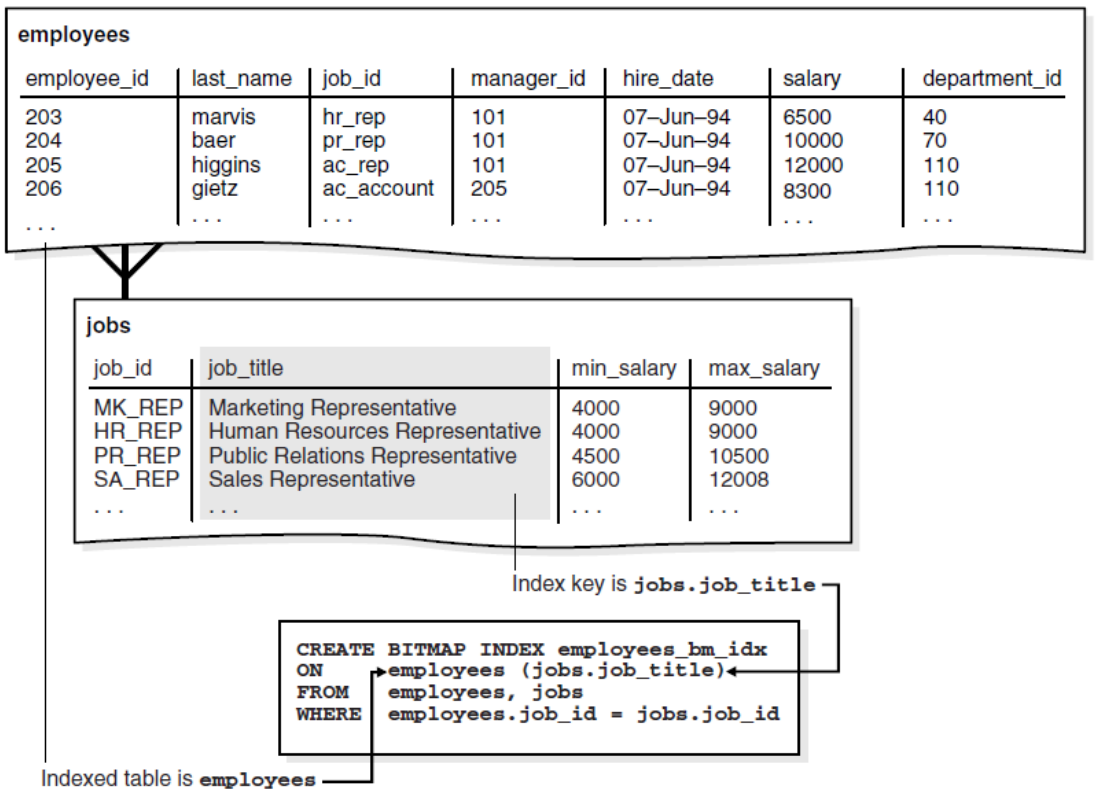
Bitmap Join Index

A bitmap join index is a bitmap index for the join of two or more tables. For each value in a table column, the index stores the rowid of the corresponding row in the indexed table.

An example when a bitmap join index would be useful:

```
SELECT COUNT(*) FROM employees, jobs
WHERE employees.job_id = jobs.job_id
AND jobs.job_title = 'Accountant';
```

Figure 3–2 Bitmap Join Index




```
jobs.job_title                      employees.rowid
----------------------------------- ------------------
Accountant                          AAAQNKAAFAAAABSAAL
Accountant                          AAAQNKAAFAAAABSAAN
Accountant                          AAAQNKAAFAAAABSAAM
Accountant                          AAAQNKAAFAAAABSAAJ
Accountant                          AAAQNKAAFAAAABSAAK
```

```
Accounting Manager                  AAAQNKAAFAAAABTAAH
Administration Assistant            AAAQNKAAFAAAABTAAC
Administration Vice President       AAAQNKAAFAAAABSAAC
Administration Vice President       AAAQNKAAFAAAABSAAB
```

Function-based Index

You can create indexes on functions and expressions that involve one or more columns in the table being indexed. A function-based index computes the value of a function or expression involving one or more columns and stores it in the index. A function-based index can be either a B-tree or a bitmap index.

Index-Organized Table

An index-organized table is a table stored in a variation of a B-tree index structure. In a heap-organized table, rows are inserted where they fit. In an index-organized table, rows are stored in an index defined on the primary key for the table. Each index entry in the B-tree also stores the non-key column values.

An index-organized table stores all data in the same structure and does not need to store the rowid.

Figure 3–3 Index-Organized Table