See Also:

"Introduction to Schema Objects" to learn more about schema object types, storage, and dependencies

Tables

45

A table describes an entity such as employees.

You define a table with a table name, such as `employees`, and set of columns. In general, you give each column a name, a data type, and a width when you create the table.

A table is a set of rows. A column identifies an attribute of the entity described by the table, whereas a row identifies an instance of the entity. For example, attributes of the employees entity correspond to columns for employee ID and last name. A row identifies a specific employee.

You can optionally specify a rule, called an integrity constraint, for a column. One example is a `NOT NULL` integrity constraint. This constraint forces the column to contain a value in every row.

See Also:

- "Overview of Tables" to learn about columns and rows, data types, table storage, and table compression
- "Data Integrity" to learn about the possible types and states of constraints

Indexes

An **index** is an optional data structure that you can create on one or more columns of a table. Indexes can increase the performance of data retrieval.

When processing a request, the database can use available indexes to locate the requested rows efficiently. Indexes are useful when applications often query a specific row or range of rows.

Indexes are logically and physically independent of the data. Thus, you can drop and create indexes with no effect on the tables or other indexes. All applications continue to function after you drop an index.

See Also:

"Introduction to Indexes" to learn about the purpose and types of indexes

45

for each row based on the time_id value according to the rules specified in the PARTITION BY RANGE clause. The range partition key value determines the non-inclusive high bound for a specified partition.

Interval Partitioning

Interval partitioning is an extension of range partitioning.

If you insert data that exceeds existing range partitions, then Oracle Database automatically creates partitions of a specified interval. For example, you could create a sales history table that stores data for each month in a separate partition.

Interval partitions enable you to avoid creating range partitions explicitly. You can use interval partitioning for almost every table that is range partitioned and uses fixed intervals for new partitions. Unless you create range partitions with different intervals, or unless you always set specific partition attributes, consider using interval partitions.

When partitioning by interval, you must specify at least one range partition. The range partitioning key value determines the high value of the range partitions, which is called the transition point. The database automatically creates interval partitions for data with values that are beyond the transition point. The lower boundary of every interval partition is the inclusive upper boundary of the previous range or interval partition. Thus, in Example 4-2, value 01-JAN-2011 is in partition p2.

The database creates interval partitions for data beyond the transition point. An interval partition extends range partitioning by instructing the database to create partitions of the specified range or interval. The database automatically creates the partitions when data inserted into the table exceeds all existing range partitions. In Example 4-2, the p3 partition contains rows with partitioning key time_id values greater than or equal to 01-JAN-2013.

Example 4-2 Interval Partitioning

Assume that you create a sales table with four partitions of varying widths. You specify that above the transition point of January 1, 2013, the database should create partitions in one month intervals. The high bound of partition p3 represents the transition point. Partition p3 and all partitions below it are in the range section, whereas all partitions above it fall into the interval section.

```
CREATE TABLE interval_sales
    ( prod_id        NUMBER(6)
    , cust_id        NUMBER
    , time_id        DATE
    , channel_id     CHAR(1)
    , promo_id       NUMBER(6)
    , quantity_sold  NUMBER(3)
    , amount_sold    NUMBER(10,2)
    )
  PARTITION BY RANGE (time_id)
  INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
    ( PARTITION p0 VALUES LESS THAN (TO_DATE('1-1-2010', 'DD-MM-YYYY'))
    , PARTITION p1 VALUES LESS THAN (TO_DATE('1-1-2011', 'DD-MM-YYYY'))
    , PARTITION p2 VALUES LESS THAN (TO_DATE('1-7-2012', 'DD-MM-YYYY'))
    , PARTITION p3 VALUES LESS THAN (TO_DATE('1-1-2013', 'DD-MM-
YYYY')) );
```

column. Specify the cluster key when creating the table cluster and when creating every table added to the table cluster.

cold buffer

A buffer in the database buffer cache that has not been recently used.

column

Vertical space in a table that represents a domain of data. A table definition includes a table name and set of columns. Each column has a name and data type.

columnar format

45 The column-based format for objects that reside in the In-Memory Column Store. The columnar format contrasts with the row format that the database uses to store objects in the database buffer cache and in data files.

commit

Action that ends a database transaction and makes permanent all changes performed in the transaction.

commit cleanout

The automatic removal of lock-related transaction information (ITL entry) from the blocks after a commit. The database removes the ITL entry only if modified blocks containing data from the committed transaction are still in the SGA, and if no other session is modifying them.

common role

A role that exists in all containers in a multitenant container database (CDB).

complete refresh

An execution of the query that defines a materialized view. A complete refresh occurs when you initially create the materialized view, unless the materialized view references a prebuilt table, or you define the table as `BUILD DEFERRED`.

composite database operation

Activity between two points in time in a single database session.