Oracle 11g - SQL

Creating Other Schema Objects

Objectives

After completing this lesson, you should be able to do the following:

- Create, maintain, and use sequences
- Create and maintain indexes
- Types of Indexes: Normal (B-Tree), BITMAP, Partitioned & Function-based Index
- Impact on Dropping of Index
- When to Drop of Index, If necessary
- Create private and public synonyms

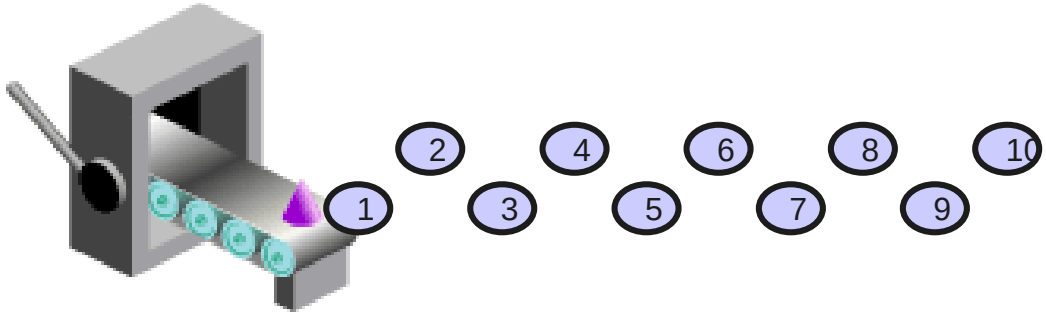
Database Objects

Object	Description
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

Sequences

A sequence:

- Can automatically generate unique numbers
- Is a sharable object
- Can be used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory



CREATE SEQUENCE Statement: Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE sequence
       [INCREMENT BY n]
       [START WITH n]
       [{MAXVALUE n | NOMAXVALUE}]
       [{MINVALUE n | NOMINVALUE}]
       [{CYCLE | NOCYCLE}]
       [{CACHE n | NOCACHE}];
```

Creating a Sequence

- Create a sequence named `DEPT_DEPTID_SEQ` to be used for the primary key of the `DEPARTMENTS` table.
- Do not use the `CYCLE` option.

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 10
                START WITH 120
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
CREATE SEQUENCE succeeded.
```

NEXTVAL and CURRVAL Pseudocolumns

- NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

Using a Sequence

- Insert a new department named “Support” in location ID 2500:

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);
1 row created.
```

- View the current value for the DEPT_DEPTID_SEQ sequence:

```
SELECT   dept_deptid_seq.CURRVAL
FROM     dual;
```

Caching Sequence Values

- Caching sequence values in memory gives faster access to those values.
- Gaps in sequence values can occur when:
  - A rollback occurs
  - The system crashes
  - A sequence is used in another table

Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option:

```
ALTER SEQUENCE dept_deptid_seq
               INCREMENT BY 20
               MAXVALUE 999999
               NOCACHE
               NOCYCLE;
ALTER SEQUENCE dept_deptid_seq succeeded.
```

Guidelines for Modifying a Sequence

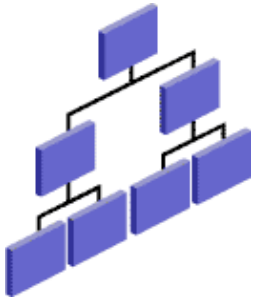
- You must be the owner or have the `ALTER` privilege for the sequence.
- Only future sequence numbers are affected.
- The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed.
- To remove a sequence, use the `DROP` statement:

```
DROP SEQUENCE dept_deptid_seq;
DROP SEQUENCE dept_deptid_seq succeeded.
```

Indexes

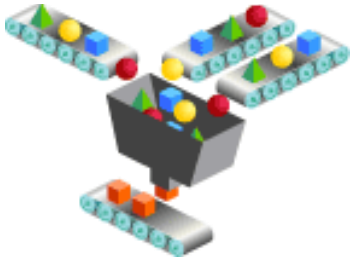
What is index:

- Is a schema object
- Can be used by the Oracle server to speed up the retrieval of rows by using a pointer
- Can reduce disk I/O by using a rapid path access method to locate data quickly
- Is independent of the table that it indexes
- Is used and maintained automatically by the Oracle server



How Are Indexes Created?

- Automatically: A unique index is created automatically when you define a `PRIMARY KEY` or `UNIQUE` constraint in a table definition.



- Manually: Users can create nonunique indexes on columns to speed up access to the rows.

Creating an Index

- Create an index on one or more columns:

```
CREATE INDEX index
ON table (column[, column]...);
```

- Improve the speed of query access to the LAST_NAME column in the EMPLOYEES table:

```
CREATE INDEX  emp_last_name_idx
ON            employees(last_name);
CREATE INDEX succeeded.
```

Types of Index

- **Normal indexes**: By default, Oracle Database creates **B-tree** indexes.
- **Bitmap indexes**: which store rowids associated with a key value as a bitmap – User-Defined

B-Tree Index

- B-trees, short for balanced trees, are the most common type of database index.
- These indexes are the standard index type. They are excellent for primary key and highly-selective indexes.
- 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
- A B-tree index is balanced because all leaf blocks automatically stay at the same depth.

Working Example: on B-TREE

- B-Tree Indexes are genrated by default when Primary Key Constraints are Created
- End-User also can create Index manually through:

  Syntex: *CREATE* *[UNIQUE] INDEX index_name ON table_name(column_name[, column_name ...]) TABLESPACE tab_space;*

  CREATE *INDEX i_emp_name ON emp(ename);*
- We can enforce uniqueness of column values using a using a unique keyword

  CREATE *UNIQUE INDEX i_employee_no ON emp(empno); nique index.*
- You can also create a composite index on multiple columns.

  CREATE *INDEX i_employees_no_name ON emp(ename, empno);*

BIMAP Index

- index is to be created with a **bitmap** for each **distinct key**, rather than indexing each row separately.
- Bitmap indexes store the rowids associated with a key value as a bitmap.
- Each bit in the bitmap corresponds to a possible rowid.
- 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 indexes are primarily designed for data warehousing

Working Example: on BITMAP

- A **bitmap index** is a type of index that uses a string of bits to quickly locate rows in a table. Bitmap indexes are normally used to index low cardinality columns in DWH
- Bitmap Index:

  CREATE BITMAP INDEX emp_bitmap_idx ON big_emp(sex);

- Bitmap Join Index:

  CREATE BITMAP INDEX emp_dept_loc ON emp(dept.loc) FROM emp, dept WHERE emp.deptno = dept.deptno TABLESPACE index_ts1;

Index Creation Guidelines

	Create an index when:
✓	A column contains a wide range of values
✓	A column contains a large number of null values
✓	One or more columns are frequently used together in a `WHERE` clause or a join condition
✓	The table is large and most queries are expected to retrieve less than 2% to 4% of the rows in the table
	Do not create an index when:
✗	The columns are not often used as a condition in the query
✗	The table is small or most queries are expected to retrieve more than 2% to 4% of the rows in the table
✗	The table is updated frequently
✗	The indexed columns are referenced as part of an expression

Impact on Creating Index

- In general, creating an index on a columns should check the impact on system or Database:
- It is sensible that all indexes on the tables in the query, are examined to determine whether they could be used.
- Obviously keeping several different indices has a negative impact on insert and delete performance.
- How about query performance: Does it make sense at all keeping too many indices on a table?
- Will the query performance improve in any case with an index added or is it even possible that the query performance will degrade with too many indices because it becomes necessary to consult all the indices to get the result?

Removing an Index

- Remove an index from the data dictionary by using the `DROP INDEX` command:

```
DROP INDEX index;
```

- Remove the `UPPER_LAST_NAME_IDX` index from the data dictionary:

```
DROP INDEX emp_last_name_idx;
DROP INDEX emp_last_name_idx succeeded.
```

- To drop an index, you must be the owner of the index or have the `DROP ANY INDEX` privilege.

Impact on Dropping of Indexes

- When you drop an index, all extents of the index segment are returned to the containing tablespace and become available for other objects in the tablespace.
- How you drop an index depends on whether you created the index explicitly with a CREATE INDEX statement, or implicitly by defining a key constraint on a table
- In some Scenarios, you cannot drop only the index associated with an enabled UNIQUE key or PRIMARY KEY constraint. To drop a constraints associated index, you must disable or drop the constraint itself.

When to Drop of Index, If necessary

Some reasons for dropping an index include:

- The index is no longer required.
- The index is not providing anticipated performance improvements for queries issued against the associated table. For example, the table might be very small, or there might be many rows in the table but very few index entries.
- Applications do not use the index to query the data.
- The index has become invalid and must be dropped before being rebuilt.
- The index has become too fragmented and must be dropped before being rebuilt.

Synonyms

Simplify access to objects by creating a synonym (another name for an object). With synonyms, you can:

- Create an easier reference to a table that is owned by another user
- Shorten lengthy object names

```
CREATE [PUBLIC] SYNONYM synonym
FOR    object;
```

Creating and Removing Synonyms

- Create a shortened name for the DEPT_SUM_VU

```
CREATE SYNONYM  d_sum
FOR  dept_sum_vu;
CREATE SYNONYM succeeded.
```

```
DROP SYNONYM d_sum;
DROP SYNONYM succeeded.
```