Text & Original Presentations: Joan Casteel

Oracle™ SQL Programming

Chapter 6

Additional Database Objects
Agenda

- Comments on, and solution, of Ch04 assignment
- Chapter 6 lecture, part 1
- Break
- Chapter 6 lecture, part 2
- In–Class assignment
Table names are stored in UPPERCASE.
- This will create no rows, since there is no match with lowercase characters in table names

Assignment statement:
- You must display contents of user_constraints table after each question, to show the changes in the constraints!

Screen shots size!!!
- Abid Qureshi & Kimberly

In-Class & Project file
- Two separate files!!!

LATE assignments
- Will NOT be graded
Chapter Objectives

- **Sequences:**
  - Describe **purpose** of a sequence and how it can be **used** by an organization
  - Explain why **gaps** may appear in the integers generated by a sequence
  - Using **CREATE SEQUENCE** command to create a sequence
  - Identify **ALTER SEQUENCE** command options that can’t be changed
  - Use **NEXTVAL** and **CURRVAL** in an **INSERT** command

- **Indexes:**
  - Explain when Oracle will automatically create an index
  - **Create** an index, using the **CREATE INDEX** command
  - **Delete** an index, using the **DELETE INDEX** command

- **Synonyms:**
  - **Create/Delete** a **PUBLIC** synonym

- **Views:**
  - Identify the contents of different versions of views used to access the data dictionary
Anything that has a **name** and defined **structure**

Usually belongs to the same user

Includes:

- **Tables** – stores **data**
- **Views** – saved **queries**
- **Sequence** – generate sequential **integers**
- **Index** – quickly **locate** specific records
- **Synonym** – **alias** for other database objects

More Objects

- **Schemas**
- **Clusters**
- **Procedures**
- **Functions**
- **Packages**
- **Triggers**
- ...
Sequences

- Used for *internal control purposes*
- Also called “Surrogate key”
  - No correlation with actual row contents
- Used to generate an integer *number sequence*
  - Useful when you need to create a unique number to act as a primary key
- User created
- Incremented & decremented using *internal Oracle routine*

**Advantages:**
- Reduces amount of application code
  - => Time efficient
- Generated & stored independent of any table
  - => The same sequence can be used for multiple tables
CREATE SEQUENCE Command

- Syntax:
  - SQL Reference

```sql
CREATE SEQUENCE sequencename
[ INCREMENT BY value ]
[ START WITH value ]
[ { MAXVALUE value | NOMAXVALUE } ]
[ { MINVALUE value | NOMICNVALUE } ]
[ { CYCLE | NOCYCLE } ]
[ { ORDER | NOORDER } ]
[ { CACHE value | NOCACHE } ];
```

- Naming conventions
  - Use "_seq" at the end of the sequence name
CREATE SEQUENCE Command

- **INCREMENT BY** value
  - Specify the **interval** between sequence number
  - Default = 1
  - If the value is negative, the sequence will be in descending order

- **START WITH** value
  - Specify the **first sequence number** to be generated
  - Default = 1

- **MAXVALUE** value
  - Specifies the **maximum** value that the sequence generates

- **NOMAXVALUE**
  - Default maximum value = \(10^{27}\) for ascending & \(-1\) for descending
CREATE SEQUENCE Command

- **MINVALUE** value
  - Specifies the minimum value that the sequence generates

- **NOMINVALUE**
  - Default minimum value = 1 for ascending & \(-10^{26}\) for descending

- **CYCLE / NOCYCLE**
  - Specifies whether the sequence will cycle after reaching its maximum / minimum value
  - You shouldn’t use CYCLE option if the sequence is used for a primary key
CREATE SEQUENCE Command

- **ORDER**
  - Guarantee that sequence numbers are generated in order of request
  - Not important for sequences used to generate primary keys
  - Useful if you are using the sequence numbers as timestamps (if you are using Oracle with Real Application Clusters)

- **NOORDER**
  - If you do not want to guarantee sequence numbers are generated in order of request. This is the default

- **CACHE / NOCACHE**
  - Specifies how many values the Oracle server will pre-allocate and keep in memory
  - Improves efficiency – faster access
  - Default = 20
CREATE SEQUENCE Example

```sql
CREATE SEQUENCE orders_order#_seq
    INCREMENT BY 1
    START WITH 1021
    NOCACHE
    NOCYCLE;
```

CREATE SEQUENCE succeeded.
Checking Sequence Existence

- To check the existence of the sequence in the `USER_OBJECTS` table in the data dictionary
- Example:

```sql
SELECT object_name
FROM user_objects
WHERE object_name LIKE '%ORDERS_%';
```

<table>
<thead>
<tr>
<th>OBJECT_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDERS_ORDERNUMBER_SEQ</td>
</tr>
</tbody>
</table>
Verifying Sequence Values

- Query **USERSEQUENCES** data dictionary view
  - Note: you might have more sequences listed on your machine
- To check the details of your created sequences

```
SELECT sequence_name, max_value, min_value, increment_by, last_number
FROM user_sequences;
```

![SQL Query and Results](image)
Pseudo Columns

**Pseudo Column**
- An Oracle assigned value (pseudo-field) used in the same context as an Oracle Database column, but not stored on disk
- Are not actual columns in a table but behave like columns
- You can select values from a pseudocolumn but you cannot insert into, update, or delete from a pseudocolumn

**Examples:**
- `SYSDATE`, `SYSTIMESTAMP`, `ROWID`, `ROWNUM`, `CURRVAL`, `NEXTVAL`, etc.
Using Sequence Values

- **NEXTVAL**
  - Pseudo column used to get successive sequence numbers
  - Must be qualified with the sequence name
  - Generates the next available sequence integer
  - Returns unique value each time it is referenced, even by other users

- **CURRVAL**
  - Pseudo column to reference the sequence number just generated by **NEXTVAL**
  - Must be qualified with the sequence name

- A sequence must be generated before we can use **NEXTVAL & CURRVAL** on that sequence
Using Sequence Values

- **NEXTVAL** – generates integer

```sql
INSERT INTO orders (order#, customer#, orderdate, shipdate, shipstreet,
                    shipcity, shipstate, shipzip)
VALUES orders order# seq.NEXTVAL, 1010, '06-APR-09', NULL, '123 WEST MAIN',
       'ATLANTA', 'GA', 30418);

1 row(s) inserted
```
Using Sequence Values (continued)

- **CURRVAL** – contains last integer generated by **NEXTVAL**

```sql
INSERT INTO orderitems (order#, item#, isbn, quantity, paideach) VALUES (orders_order#_seq.CURRVAL, 1, 8117949391, 1, 8.50);
```

1 rows inserted
Using Cache for Sequences

- Caching sequence values in memory provide faster access to those values & improves the overall efficiency.
- The cache is populated the first time you refer to the sequence.
- Each request for a `NEXTVAL` is retrieved from the cache until the last value is used, then another sequence is populated the cache in the memory.
Caution Using Sequence Cache !!

- If the sequence is not cached, you can view the next sequence value, without incrementing it, using `USERSEQUENCES` table.
- If the sequence is cached, you can’t find out what is the next value, without actually using the sequence value.
- If the numbers in cache are not used, they are wasted (Cannot be returned for later use).
Gaps in sequence values can occur when the following occurs:

- A rollback
- System crash
- Sequence used in another table or user
Using Sequence Values

- Syntax:
  - `sequencename.NEXTVAL`
  - `sequencename.CURRVAL`

- If the field represents a primary key and the number already exists in the table, an error message will be generated.
Rules for NEXTVAL & CURRVAL

- When to use NEXTVAL & CURRVAL?
  - The SELECT list of a SELECT statement that is not part of a subquery
  - The SELECT list of a subquery in an INSERT statement
  - The VALUES clause of an INSERT statement
  - The SET clause of an UPDATE statement

- When NOT to use NEXTVAL & CURRVAL?
  - A SELECT list of a view
  - A SELECT statement with the DISTINCT keyword
  - A SELECT statement with GROUP BY, HAVING, or UPDATE statement
  - The DEFAULT expression in a CREATE TABLE or ALTER TABLE statements
### Altering Sequence Setting

**Syntax:**

```sql
ALTER SEQUENCE sequencename
[INCREMENT BY value]
[{{MAXVALUE value | NOMAXVALUE}}]
[{{MINVALUE value | NOMINVALUE}}]
[{{CYCLE | NOCYCLE}}]
[{{ORDER | NOORDER}}]
[{{CACHE value | NOCACHE}}];
```

- Which clauses are missing compared to **CREATE SEQUENCE** command?
Altering Sequence Guidelines

- **START WITH** value cannot be altered
  - To restart at different number, you **must drop** the sequence then re-create it
- Changes **cannot make current integers invalid**
- If the maximum value is already reached, an error message is generated indicating that the sequence exceeds the maximum value
- You must be the **sequence owner** to have alter privilege for a sequence
- Only future sequence numbers are in effect
ALTER SEQUENCE Example

```
ALTER SEQUENCE orders_order#_seq
  INCREMENT BY 10;
```

```
ALTER SEQUENCE orders_order#_seq succeeded.
```
Removing a Sequence

- Use the **DROP SEQUENCE** command to delete a sequence
- You **must be the owner** of the sequence to be able to drop it
- Previous values generated are **not affected** by removing a sequence from a database
- Once the sequence is removed from the data dictionary, you can’t reference it

```
DROP SEQUENCE orders_order#_seq;
```

Results: `DROP SEQUENCE orders_order#_seq succeeded.`
What is an Index?

- A database object, **based on table values**
- Stores a **frequently referenced value** and the **ROWID** of the record containing the value
- Similar to a book index
- Provides **fast access/search** to the rows of a table through minimizing disk I/O
  - If you don’t have an index on a column, a **full table scan** will occur
- Can be **based on one column**, multiple columns, **functions**, or **expressions**
- **Independent** of the table that it indexes
  - Can be created or dropped without affecting the table or other indexes
- If you drop the table, the corresponding indexes are dropped too
- Managing indexes is usually the responsibility of a DBA
Creating Indexes

1. **Implicitly** (automatically) by Oracle:
   - Whenever a column is referenced by **PRIMARY KEY** or **UNIQUE** constraints
   - Allows Oracle to determine whether a value exists in a table without having to perform a full table scan
   - Name of the index is that of the constraint
   - Recommended

2. **Explicitly** (manually) by a user
   - To speed up retrieval of rows based on a column that is frequently referenced in a **WHERE** clause
   - Example:
     - You can create a **FOREIGN KEY** column index for a join in query to improve performance
Index Guidelines

- Whenever the indexed column(s) is used as a search condition, Oracle will automatically use the index instead of a full search.

- Every time a DML operation is performed on the underlying table, the index is automatically updated, which might slow down the operation, if the update is frequent.

- Indexes are more beneficial only if a small percentage (<=5%) of the table is expected to be returned in query results.
Index Rules

- **When to use an index?**
  - Table is large & a specific column is frequently used
  - Column contains a large number of NULL values
  - Column contains wide range of values
  - Most queries return less than 5% of the rows of the table

- **When **NOT** to use an index?**
  - When the table is small
  - Columns are not often used as a condition in a query
  - If queries return more than 5% of the rows of the table
  - Table is updated more often
  - Indexed column are used as part of expressions
Indexes Types

1) Binary tree (\texttt{BTree}) index:

2) Bitmap index:

http://www.dba-oracle.com/oracle_tips_bitmapped_indexes.htm

- SQL reference on indexes
- More in the following links:
  - TechRepublic.com
Binary Tree (B–Tree) Indexes

- Default, For columns containing many distinct values
- Very efficient for searching
  - Whenever an indexed column is involved in a comparison using the operators: <, <=, =, >=, >
- Implicitly create an index by PRIMARY KEY and UNIQUE constraints
- Explicitly create an index by using the CREATE INDEX command
B–Tree Index Example

B-tree index

Branch blocks

Leaf blocks

Table

Search for zip code of 90404
CREATE INDEX Command Examples

Enter SQL Statement:

```
CREATE INDEX customers_zip_idx
ON customers (zip);
```

CREATE INDEX succeeded.

Enter SQL Statement:

```
CREATE INDEX customer_name_idx
ON customers (lastname, firstname);
```

CREATE INDEX succeeded.
The Explain Plan

```
SELECT customer#, lastname, city, state, zip
FROM customers
WHERE zip = 49006;
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Optimizer</th>
<th>Cost</th>
<th>Cardinality</th>
<th>Bytes</th>
<th>Partition Start</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT STATEMENT</td>
<td>ALL_ROWS</td>
<td>3</td>
<td>1</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>TABLE ACCESS(FULL) SCOTT.CUSTOMERS</td>
<td>3</td>
<td>1</td>
<td>35</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Full table scan performed; index ignored.
Bitmap Indexes

- A bitmap for each key value is used instead of a list of ROWID’s.
- Each bit in the bitmap corresponds to a possible ROWID:
  - A set(1) bit means that the row with the corresponding ROWID contains the key value.
  - A mapping function converts the bit position to an actual ROWID.
- Stores the location of data rows based sequences of 0's and 1's.
- Requires much less storage than a BTree.
- Applicable to read-only type environments.
- For columns containing only few values.
- Used in data warehousing applications.
Bitmap Index Example

```
CREATE BITMAP INDEX customers_region_idx
ON customers (region);
```

CREATE BITMAP succeeded.

Bitmap index
Region

<table>
<thead>
<tr>
<th>N</th>
<th>NW</th>
<th>NE</th>
<th>S</th>
<th>SE</th>
<th>SW</th>
<th>W</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Function-Based Indexes

- If the query is based on calculated value
- Can only be used if you have DBA privileges

```sql
CREATE INDEX books_profit_idx
    ON books (retail-cost);

CREATE INDEX orders_shipdate_idx
    ON orders(NVL(shipdate,'null'));
```
Index Organized Tables (IOT)

- An **IOT** stores table contents in a B-tree index structure.
- Use the **“ORGANIZATION INDEX”** option in a **CREATE TABLE** statement to build an IOT.

```sql
CREATE TABLE books2
(ISBN VARCHAR2(10),
title VARCHAR2(30),
pubdate DATE,
pubID NUMBER (2),
cost NUMBER (5,2),
retail NUMBER (5,2),
category VARCHAR2(12),
    CONSTRAINT books2_isbn_pk PRIMARY KEY(isbn))
ORGANIZATION INDEX;
```
Verifying an Index Example

- Use the `USER_INDEXES` data dictionary view to determine that the index exists.

```sql
SELECT table_name, index_name, index_type
FROM user_indexes
WHERE table_name = 'CUSTOMERS';
```

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>INDEX_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMERS</td>
<td>CUSTOMERS_REGION_IDX</td>
<td>BITMAP</td>
</tr>
<tr>
<td>CUSTOMERS</td>
<td>CUSTOMER_NAME_IDX</td>
<td>NORMAL</td>
</tr>
<tr>
<td>CUSTOMERS</td>
<td>CUSTOMERS_ZIP_IDX</td>
<td>NORMAL</td>
</tr>
<tr>
<td>CUSTOMERS</td>
<td>CUSTOMERS_CUSTOMER##_PK</td>
<td>NORMAL</td>
</tr>
</tbody>
</table>
USER_IND_COLUMNS

```
SELECT table_name, index_name, column_name
FROM user_ind_columns
WHERE table_name = 'CUSTOMERS';
```
Removing Indexes

- Use **DROP INDEX** command to remove an index

![Screen shot of SQL COMMAND showingDROP INDEX books_profitcalc_idx; succeeded.](image)
Synonyms

- Serve as **permanent aliases** for database objects
- Simplify object references

**Private synonym**
- Default – When an object is created by a user, unless otherwise specified, it will be private (belong to the schema of that user)
- Only **available to user who created them**
- Name must be distinct from all other objects owned by the same user

**Public synonym**
- Available to **all database users**

**Schema:**
- A **collection of objects**
- Multiple objects with the same name can exist in the database if they belong to different schemas
- To reference an object, we should prefix the table name with the schema name, or use synonym.
CREATE SYNONYM Command

- Only if you have DBA privileges
- Syntax:

  CREATE [PUBLIC] SYNONYM synonymname
  FOR objectname;
CREATE SYNONYM Example

```sql
CREATE SYNONYM orderentry FOR orders;
```

CREATE SYNONYM succeeded.

Synonym used in place of table name

```sql
SELECT *
FROM orderentry
WHERE customer# = 1010;
```

Results:

<table>
<thead>
<tr>
<th>ORDER#</th>
<th>CUSTOMER#</th>
<th>ORDERDATE</th>
<th>SHIPDATE</th>
<th>SHIPSTREET</th>
<th>SHIPCITY</th>
<th>SHIPSTATE</th>
<th>SHIPZIP</th>
<th>SHIPCOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1001</td>
<td>1010 31-MAR-09</td>
<td>01-APR-09</td>
<td>114 EAST SAVANNAH</td>
<td>ATLANTA</td>
<td>GA</td>
<td>30314</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1011</td>
<td>1010 03-APR-09</td>
<td>05-APR-09</td>
<td>102 WEST LAFAYETTE</td>
<td>ATLANTA</td>
<td>GA</td>
<td>30311</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1021</td>
<td>1010 05-APR-09</td>
<td>(null)</td>
<td>123 WEST MAIN</td>
<td>ATLANTA</td>
<td>GA</td>
<td>30418</td>
<td>(null)</td>
</tr>
</tbody>
</table>
Deleting a SYNONYM

- A private synonym can be deleted by owner
- A **PUBLIC** synonym can only be deleted by a user with DBA privileges
- Syntax:
  
  \[
  \text{DROP [PUBLIC] SYNONYM synonymname;}
  \]
Deleting a SYNONYM example

```
DROP SYNONYM orderentry;
```

```
DROP SYNONYM orderentry succeeded.
```
Data Dictionary

- Owned by the user **SYS** that is automatically created by Oracle
- Includes information about:
  - Object's **name**, **type**, **structure**, and **owner**
  - Identify the **users** who have access to each object
- Can't be queried directly, but accessed through a set of **predefined views**
  - (e.g. **USER_INDEXES**, **USER_OBJECTS**, ...).
Data Dictionary Prefixes

- **USER_**
  - Prefix for objects contained by the user

- **ALL_**
  - Objects accessible by the user, even if they aren't owned by the user

- **DBA_**
  - All objects within the database (e.g. `DBA_CONSTRAINTS`)

- **V$**
  - Dynamic views to access statistics relations to database performance
Summary

- Sequences generate series of integers that can be stored in any table.
- Gaps may occur if values are stored in various tables, if numbers are cached but not used, or if a rollback occurred.
- We generate & manipulate sequences with `NEXTVAL` & `CURRVAL` pseudo-columns.
- Indexes speed up query process and improve query performance on large tables if <= 5% rows retrieved.
- Oracle 9i automatically creates indexes for `PRIMARY KEY` & `UNIQUE` constraints.
- Synonyms provides aliases for database objects.
- Only users with DBA privileges can drop public synonyms or create indexes with functions.
- Use prefixes to objects to specify the owner (`USER_`, `ALL_`, `DBA_`, `V$`).