ORACLE®

ACADEMY

Database Programming with SQL

15-1
Creating Views

View

- A view, like a table, is a database object.
- However, views are not "real" tables.
- They are logical representations of existing tables or of another view.
- Views contain no data of their own.
- They function as a window through which data from tables can be viewed or changed.

View

- The tables on which a view is based are called "base" tables.
- The view is a query stored as a SELECT statement in the data dictionary.

```
CREATE VIEW view_employees
AS SELECT employee_id,first_name, last_name, email
   FROM employees
   WHERE employee_id BETWEEN 100 and 124;
```

```
SELECT *
FROM view_employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL
100	Steven	King	SKING
101	Neena	Kochhar	NKOCHHAR
102	Lex	De Haan	LDEHAAN
124	Kevin	Mourgos	KMOURGOS
103	Alexander	Hunold	AHUNOLD
104	Bruce	Ernst	BERNST
107	Diana	Lorentz	DLORENTZ

Why Use Views?

- Views restrict access to base table data because the view can display selective columns from the table.
- Views can be used to reduce the complexity of executing queries based on more complicated SELECT statements.
- For example, the creator of the view can construct join statements that retrieve data from multiple tables.
- The user of the view neither sees the underlying code nor how to create it.
- The user, through the view, interacts with the database using simple queries.

Why Use Views?

- Views can be used to retrieve data from several tables, providing data independence for users.
- Users can view the same data in different ways.
- Views provide groups of users with access to data according to their particular permissions or criteria.

Creating a View

- To create a view, embed a subquery within the CREATE VIEW statement.
- The syntax of a view statement is as follows:

```
CREATE [OR REPLACE] [FORCE| NOFORCE] VIEW view  [(alias [,
   alias]...)] AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

Creating a View

OR REPLACE	Re-creates the view if it already exists.
FORCE	Creates the view whether or not the base tables exist.
NOFORCE	Creates the view only if the base table exists (default).
view_name	Specifies the name of the view.
alias	Specifies a name for each expression selected by the view's query.
subquery	Is a complete SELECT statement. You can use aliases for the columns in the SELECT list. The subquery can contain complex SELECT syntax.

Creating a View

WITH CHECK OPTION	Specifies that rows remain accessible to the view after insert or update operations.
CONSTRAINT	Is the name assigned to the CHECK OPTION constraint.
WITH READ ONLY	Ensures that no DML operations can be performed on this view.

Creating a View

- Example:

```
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id, region_id, country_name, capitol
   FROM wf_countries
   WHERE location LIKE '%Europe';
```

```
SELECT * FROM view_euro_countries
ORDER BY country_name;
```

COUNTRY_ID	REGION_ID	COUNTRY_NAME	CAPITOL
22	155	Bailiwick of Guernsey	Saint Peter Port
203	155	Bailiwick of Jersey	Saint Helier
387	39	Bosnia and Herzegovina	Sarajevo
420	151	Czech Republic	Prague
298	154	Faroe Islands	Torshavn
49	155	Federal Republic of Germany	Berlin
33	155	French Republic	Paris
…	…	…	…

Guidelines for Creating a View

- The subquery that defines the view can contain complex SELECT syntax.
- For performance reasons, the subquery that defines the view should not contain an ORDER BY clause. The ORDER BY clause is best specified when you retrieve data from the view.
- You can use the OR REPLACE option to change the definition of the view without having to drop it or re-grant object privileges previously granted on it.
- Aliases can be used for the column names in the subquery.

CREATE VIEW Features

- Two classifications of views are used: simple and complex.
- The table summarizes the features of each view.

Feature	Simple Views	Complex Views
Number of tables used to derive data	One	One or more
Can contain functions	No	Yes
Can contain groups of data	No	Yes
Can perform DML operations (INSERT, UPDATE, DELETE) through a view	Yes	Not always

Simple View

- The view shown below is an example of a simple view.
- The subquery derives data from only one table and it does not contain a join function or any group functions.
- Because it is a simple view, INSERT, UPDATE, DELETE, and MERGE operations affecting the base table could possibly be performed through the view.

```
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id, country_name, capitol
   FROM wf_countries
   WHERE location LIKE '%Europe';
```

Simple View

- Column names in the SELECT statement can have aliases as shown below.
- Note that aliases can also be listed after the CREATE VIEW statement and before the SELECT subquery.

```
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id AS "ID", country_name AS "Country",
       capitol AS "Capitol City"
   FROM wf_countries
   WHERE location LIKE '%Europe';
```

```
CREATE OR REPLACE VIEW view_euro_countries("ID", "Country",
    "Capitol City")
AS SELECT country_id, country_name, capitol
   FROM wf_countries
   WHERE location LIKE '%Europe';
```

Complex View

- Complex views are views that can contain group functions and joins.
- The following example creates a view that derives data from two tables.

```
CREATE OR REPLACE VIEW view_euro_countries
   ("ID", "Country", "Capitol City", "Region")
AS SELECT c.country_id, c.country_name, c.capitol, r.region_name
   FROM wf_countries c JOIN wf_world_regions r
   USING (region_id)
   WHERE location LIKE '%Europe';
```

```
SELECT *
FROM view_euro_countries;
```

Complex View

ID	Country	Capitol City	Region
375	Republic of Belarus	Minsk	Eastern Europe
48	Republic of Poland	Warsaw	Eastern Europe
421	Slovak Republic	Bratislava	Eastern Europe
36	Republic of Hungary	Budapest	Eastern Europe
90	Republic of Turkey	Ankara	Eastern Europe
40	Romania	Bucharest	Eastern Europe
373	Republic of Moldova	Chisinau	Eastern Europe
370	Republic of Lithuania	Vilnius	Eastern Europe
371	Republic of Latvia	Riga	Eastern Europe
372	Republic of Estonia	Tallinn	Eastern Europe
…	…	…	…

Complex View

- Group functions can also be added to complex-view statements.

```
CREATE OR REPLACE VIEW view_high_pop
   ("Region ID", "Highest population")
AS SELECT region_id, MAX(population)
   FROM wf_countries
   GROUP BY region_id;
```

```
SELECT * FROM view_high_pop;
```

Region ID	Highest population
5	188078227
9	20264082
11	131859731
13	107449525
14	74777981
15	78887007
17	62660551
18	44187637
21	298444215
…	…

Database Programming with SQL

15-2
DML Operations and Views

ORACLE ACADEMY

DML Statements and Views

- The DML operations INSERT, UPDATE, and DELETE can be performed on simple views.
- These operations can be used to change the data in the underlying base tables.
- If you create a view that allows users to view restricted information using the WHERE clause, users can still perform DML operations on all columns of the view.

DML Statements and Views

- For example, the view shown on the right was created for the managers of department 50 from the employees database.
- The intent of this view is to allow managers of department 50 to see information about their employees.

```
CREATE VIEW view_dept50
AS SELECT department_id, employee_id,first_name, last_name, salary
 FROM copy_employees
 WHERE department_id = 50;
```

```
SELECT * FROM view_dept50;
```

DEPARTMENT_ID	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
50	124	Kevin	Mourgos	5800
50	141	Trenna	Rajs	3500
50	142	Curtis	Davies	3100
50	143	Randall	Matos	2600
50	144	Peter	Vargas	2500

Controlling Views

- Using the view as stated, it is possible to INSERT, UPDATE, and DELETE information for all rows in the view, even if this results in a row no longer being part of the view.
- This may not be what the DBA intended when the view was created.
- To control data access, two options can be added to the CREATE VIEW statement:
  - WITH CHECK OPTION
  - WITH READ ONLY

Views with CHECK Option

- The view is defined without the WITH CHECK OPTION.

```
CREATE VIEW view_dept50
AS SELECT department_id, employee_id, first_name, last_name, salary
   FROM copy_employees
   WHERE department_id = 50;
```

- Using the view, employee_id 124 has his department changed to dept_id 90.

```
UPDATE view_dept50
SET department_id = 90
WHERE employee_id = 124;
```

```
1 row(s) updated.
```

- The update succeeds, even though this employee is now not part of the view.

Views with CHECK Option

- The WITH CHECK OPTION ensures that DML operations performed on the view stay within the domain of the view.
- Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.
- Notice in the example below that the WITH CHECK OPTION CONSTRAINT was given the name view_dept50_check.

```
CREATE OR REPLACE VIEW view_dept50
AS SELECT department_id, employee_id, first_name, last_name, salary
   FROM employees
   WHERE department_id = 50
WITH CHECK OPTION CONSTRAINT view_dept50_check;
```

Views with CHECK Option

- Now, if we attempt to modify a row in the view that would take it outside the domain of the view, an error is returned.

```
UPDATE view_dept50
SET department_id = 90
WHERE employee_id = 124;
```

```
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Views with READ ONLY

- The WITH READ ONLY option ensures that no DML operations occur through the view.
- Any attempt to execute an INSERT, UPDATE, or DELETE statement will result in an Oracle server error.

```
CREATE OR REPLACE VIEW view_dept50
AS SELECT department_id, employee_id, first_name, last_name, salary
 FROM employees
 WHERE department_id = 50
WITH READ ONLY;
```

DML Restrictions

- Simple views and complex views differ in their ability to allow DML operations through a view.
- For simple views, DML operations can be performed through the view.
- For complex views, DML operations are not always allowed.
- The following three rules must be considered when performing DML operations on views.

DML Restrictions

- You cannot remove a row from an underlying base table if the view contains any of the following:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM Keyword

DML Restrictions

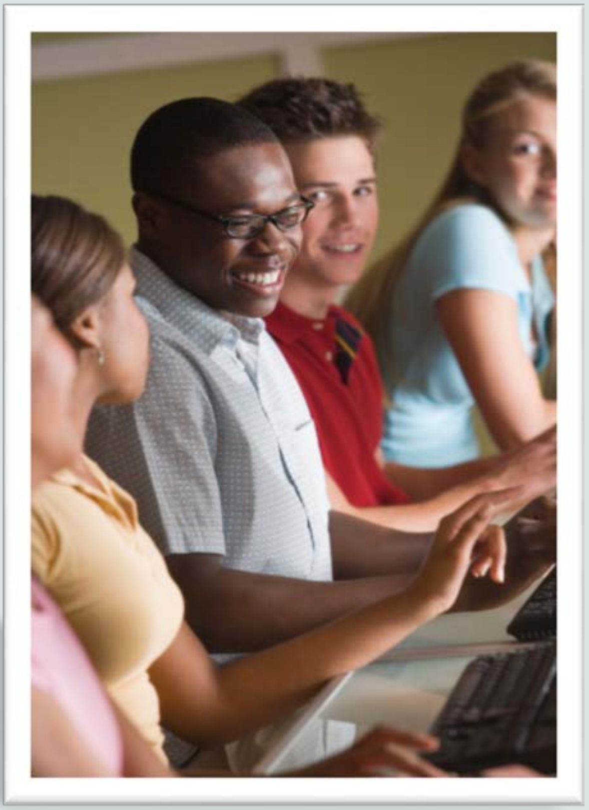
- You cannot modify data through a view if the view contains:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword
  - Columns defined by expressions

DML Restrictions

- You cannot add data through a view if the view:
  - includes group functions
  - includes a GROUP BY clause
  - includes the DISTINCT keyword
  - includes the pseudocolumn ROWNUM keyword
  - includes columns defined by expressions
  - does not include NOT NULL columns in the base tables

Database Programming with SQL

15-3
Managing Views



ORACLE ACADEMY

Deleting a View

- Because a view contains no data of its own, removing it does not affect the data in the underlying tables.
- If the view was used to INSERT, UPDATE, or DELETE data in the past, those changes to the base tables remain.
- Deleting a view simply removes the view definition from the database.

Deleting a View

- Remember, views are stored as SELECT statements in the data dictionary.
- Only the creator or users with the DROP ANY VIEW privilege can remove a view.
- The SQL syntax to remove a view is:

```
DROP VIEW viewname;
```

Inline Views

- Inline views are also referred to as subqueries in the FROM clause.
- You insert a subquery in the FROM clause just as if the subquery was a table name.
- Inline views are commonly used to simplify complex queries by removing join operations and condensing several queries into one.

Inline Views

- As shown in the example below, the FROM clause contains a SELECT statement that retrieves data much like any SELECT statement.
- The data returned by the subquery is given an alias (d), which is then used in conjunction with the main query to return selected columns from both query sources.

```
SELECT e.last_name, e.salary, e.department_id, d.maxsal
FROM employees e,
       (SELECT department_id, max(salary) maxsal
        FROM employees
        GROUP BY department_id) d
WHERE e.department_id = d.department_id
AND e.salary = d.maxsal;
```

TOP-N-ANALYSIS

- Top-n-analysis is a SQL operation used to rank results.
- The use of top-n-analysis is useful when you want to retrieve the top 5 records, or top-n records, of a result set returned by a query.

```
SELECT ROWNUM AS "Longest employed", last_name, hire_date
FROM employees
WHERE ROWNUM <=5
ORDER BY hire_date;
```

Longest employed	LAST_NAME	HIRE_DATE
1	King	17-Jun-1987
4	Whalen	17-Sep-1987
2	Kochhar	21-Sep-1989
3	De Haan	13-Jan-1993
5	Higgins	07-Jun-1994

TOP-N-ANALYSIS

- The top-n-analysis query uses an inline view (a subquery) to return a result set.
- You can use ROWNUM in your query to assign a row number to the result set.
- The main query then uses ROWNUM to order the data and return the top five.

```
SELECT ROWNUM AS "Longest employed", last_name, hire_date
FROM (SELECT last_name, hire_date
      FROM employees
      ORDER BY hire_date)
WHERE ROWNUM <=5;
```

TOP-N-ANALYSIS

Longest employed	LAST_NAME	HIRE_DATE
1	King	17-Jun-1987
2	Whalen	17-Sep-1987
3	Kochhar	21-Sep-1989
4	Hunold	03-Jan-1990
5	Ernst	21-May-1991

- In the example above, the inline view first selects the list of last_names and hire_dates of the employees:

```
(SELECT last_name, hire_date FROM employees...
```

- Then the inline view orders the years from oldest to newest.

```
...ORDER BY hire_date)
```

TOP-N-ANALYSIS

- The outer query WHERE clause is used to restrict the number of rows returned and must use a < or <= operator.

```
SELECT ROWNUM AS "Longest employed", last_name, hire_date
FROM (SELECT last_name, hire_date
      FROM employees
      ORDER BY hire_date)
WHERE ROWNUM <=5;
```

Longest employed	LAST_NAME	HIRE_DATE
1	King	17-Jun-1987
2	Whalen	17-Sep-1987
3	Kochhar	21-Sep-1989
4	Hunold	03-Jan-1990
5	Ernst	21-May-1991

Database Programming with SQL

16-1
Working With Sequences

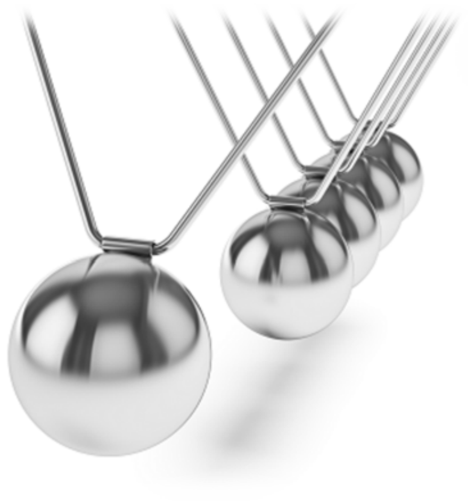
ORACLE ACADEMY

The Sequence Object

- You already know how to create two kinds of database objects, the TABLE and the VIEW.
- A third database object is the SEQUENCE.
- A SEQUENCE is a shareable object used to automatically generate unique numbers.
- Because it is a shareable object, multiple users can access it.
- Typically, sequences are used to create a primary-key value.

The Sequence Object

- As you'll recall, primary keys must be unique for each row. The sequence is generated and incremented (or decremented) by an internal Oracle routine.
- This object is a time-saver for you because it reduces the amount of code you need to write.



The Sequence Object

- Sequence numbers are stored and generated independently of tables.
- Therefore, the same sequence can be used for multiple tables.
- To create a SEQUENCE:

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

Sequence Syntax

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

sequence	is the name of the sequence generator (object)
INCREMENT BY n	specifies the interval between sequence numbers where n is an integer (If this clause is omitted, the sequence increments by 1.)
START WITH n	specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)

Sequence Syntax

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

MAXVALUE n	specifies the maximum value the sequence can generate
NOMAXVALUE	specifies a maximum value of 10^27 for an ascending sequence and -1 for a descending sequence (default)
MINVALUE n	specifies the minimum sequence value

Sequence Syntax

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

NOMINVALUE	specifies a minimum value of 1 for an ascending sequence and –(10^26) for a descending sequence (default)
CYCLE | NOCYCLE	specifies whether the sequence continues to generate values after reaching its maximum or minimum value (NOCYCLE is the default option.)

Sequence Syntax

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

CACHE n | NOCACHE	specifies how many values the Oracle server pre-allocates and keeps in memory. (By default, the Oracle server caches 20 values.) If the system crashes, the values are lost.

Creating a Sequence

- In the SEQUENCE created for the London Marathon runners, the numbers will increment by 1, starting with the number 1.
- In this case, beginning the sequence with 1 is probably the best starting point.

```
CREATE SEQUENCE runner_id_seq
  INCREMENT BY 1
  START WITH 1
  MAXVALUE 50000
  NOCACHE
  NOCYCLE;
```

Creating a Sequence

- It is a tradition that the best runner in the elite group wears number 1.
- For other situations, such as department IDs and employee IDs, the starting number may be assigned differently.
- Because there will be at least 30,000 runners, the sequence's maximum value was set well above the expected number of runners.

```
CREATE SEQUENCE runner_id_seq
  INCREMENT BY 1
  START WITH 1
  MAXVALUE 50000
  NOCACHE
  NOCYCLE;
```

Confirming Sequences

- To verify that a sequence was created, query the USER_OBJECTS data dictionary.
- To see all of the SEQUENCE settings, query the USER_SEQUENCES data dictionary as shown below.
- List the value names in the SELECT statement as shown below.

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

NEXTVAL and CURRVAL Pseudocolumns

- The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified sequence.
- You must qualify NEXTVAL with the sequence name.
- When you reference sequence.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.

NEXTVAL and CURRVAL Pseudocolumns

- The example below inserts a new department in the DEPARTMENTS table.
- It uses the DEPARTMENTS_SEQ sequence for generating a new department number as follows:

```
INSERT INTO departments
        (department_id, department_name, location_id)
VALUES  (departments_seq.NEXTVAL, 'Support', 2500);
```

NEXTVAL and CURRVAL Pseudocolumns

- Suppose now you want to hire employees to staff the new department.
- The INSERT statement to be executed for all new employees can include the following code:

```
INSERT INTO employees
       (employee_id, department_id, ...)
VALUES (employees_seq.NEXTVAL, dept_deptid_seq .CURRVAL, ...);
```

- Note: The preceding example assumes that a sequence called EMPLOYEES_SEQ has already been created for generating new employee numbers.

NEXTVAL and CURRVAL Pseudocolumns

- The CURRVAL pseudocolumn in the example below is used to refer to a sequence number that the current user has just generated.
- NEXTVAL must be used to generate a sequence number in the current user's session before CURRVAL can be referenced.
- You must qualify CURRVAL with the sequence name.

NEXTVAL and CURRVAL Pseudocolumns

- When sequence.CURRVAL is referenced, the last value generated by that user's process is returned.

```
INSERT INTO employees
       (employee_id, department_id, ...)
VALUES (employees_seq.NEXTVAL, dept_deptid_seq.CURRVAL, ...);
```

Using a Sequence

- After you create a sequence, it generates sequential numbers for use in your tables. Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.
- You can use NEXTVAL and CURRVAL in the following contexts:
  - 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

Using a Sequence

- You cannot use NEXTVAL and CURRVAL in the following contexts:
  - The SELECT list of a view
  - A SELECT statement with the DISTINCT keyword
  - A SELECT statement with GROUP BY, HAVING, or ORDER BY clauses
  - A subquery in a SELECT, DELETE, or UPDATE statement
  - The DEFAULT expression in a CREATE TABLE or ALTER TABLE statement

Using a Sequence

- To continue our London Marathon example, a table was created for the runners:

```
CREATE TABLE runners
(runner_id NUMBER(6,0) CONSTRAINT runners_id_pk PRIMARY KEY,
 first_name VARCHAR2(30),
 last_name VARCHAR2(30));
```

Using a Sequence

- We then create the sequence that will generate values for the runner_id primary key column.

```
CREATE SEQUENCE runner_id_seq
   INCREMENT BY 1
   START WITH 1
   MAXVALUE 50000
   NOCACHE
   NOCYCLE;
```

Using a Sequence

- Using the following syntax would allow new participants to be inserted into the runners table.
- The runner's identification number would be generated by retrieving the NEXTVAL from the sequence.

```
INSERT INTO runners
        (runner_id, first_name, last_name)
VALUES  (runner_id_seq.NEXTVAL, 'Joanne', 'Everely');
```

```
INSERT INTO runners
        (runner_id, first_name, last_name)
VALUES  (runner_id_seq.NEXTVAL, 'Adam', 'Curtis');
```

Using a Sequence

- To confirm the sequence worked correctly, we query the table:

```
SELECT runner_id, first_name, last_name
FROM runners;
```

RUNNER_ID	FIRST_NAME	LAST_NAME
1	Joanne	Everely
2	Adam	Curtis

Using a Sequence

- To view the current value for the runners_id_seq, CURRVAL is used.
- Note the use of the DUAL table in this example.
- Oracle Application Express will not execute this query, but you should understand how this works.

```
SELECT runner_id_seq.CURRVAL
FROM dual;
```

Nonsequential Numbers

- Although sequence generators issue sequential numbers without gaps, this action occurs independently of a database commit or rollback.
- Gaps (nonsequential numbers) can be generated by:
  - Rolling back a statement containing a sequence, the number is lost.
  - A system crash. If the sequence caches values into the memory and the system crashes, those values are lost.
  - The same sequence being used for multiple tables. If you do so, each table can contain gaps in the sequential numbers.

Viewing the Next Value

- If the sequence was created with NOCACHE, it is possible to view the next available sequence value without incrementing it by querying the USER_SEQUENCES table.

```
SELECT sequence_name, min_value, max_value, last_number AS "Next number"
FROM USER_SEQUENCES
WHERE sequence_name = 'RUNNER_ID_SEQ';
```

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	Next number
RUNNER_ID_SEQ	1	50000	3

Modifying a Sequence

- As with the other database objects you've created, a SEQUENCE can also be changed using the ALTER SEQUENCE statement.
- What if the London Marathon exceeded the 50,000 runner registrations and you needed to add more numbers?
- The sequence could be changed to increase the MAXVALUE without changing the existing number order.

```
ALTER SEQUENCE runner_id_seq
               INCREMENT BY 1
               MAXVALUE 999999
               NOCACHE
               NOCYCLE;
```

Modifying a Sequence

- Some validation is performed when you alter a sequence.
- For example, a new MAXVALUE that is less than the current sequence number cannot be executed.

```
ALTER SEQUENCE runner_id_seq
               INCREMENT BY 1
               MAXVALUE 90
               NOCACHE
               NOCYCLE;
```

```
ERROR at line 1:
ORA-04009: MAXVALUE cannot be made to be less than the current value
```

ALTER SEQUENCE Guidelines

- A few guidelines apply when executing an ALTER SEQUENCE statement.
- They are:
  – You must be the owner or have the ALTER privilege for the sequence in order to modify it.
  – Only future sequence numbers are affected by the ALTER SEQUENCE statement.
  – The START WITH option cannot be changed using ALTER SEQUENCE. The sequence must be dropped and re-created in order to restart the sequence at a different number.

Removing a Sequence

- To remove a sequence from the data dictionary, use the DROP SEQUENCE statement.
- You must be the owner of the sequence or have DROP ANY SEQUENCE privileges to remove it.
- Once removed, the sequence can no longer be referenced.

```
DROP SEQUENCE runner_id_seq;
```

Database Programming with SQL

16-2
Indexes and Synonyms

ORACLE ACADEMY

Indexes

- An Oracle Server index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically.
- If you do not have an index on the column you're selecting, then a full table scan occurs.
- An index provides direct and fast access to rows in a table.
- Its purpose is to reduce the necessity of disk I/O (input/output) by using an indexed path to locate data quickly.

Indexes

- The index is used and maintained automatically by the Oracle Server. Once an index is created, no direct activity is required by the user.
- A ROWID is a base 64 string representation of the row address containing block identifier, row location in the block, and the database file identifier.
- Indexes use ROWID's because they are the fastest way to access any particular row.

Indexes

- Indexes are logically and physically independent of the table they index.
- This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

COUNTRY_ID	COUNTRY_NAME	CAPITOL	REGION_ID
1	United States of America	Washington, DC	21
2	Canada	Ottawa	21
3	Republic of Kazakhstan	Astana	143
7	Russian Federation	Moscow	151
12	Coral Sea Islands Territory	-	9
13	Cook Islands	Avarua	9
15	Europa Island	-	18
20	Arab Republic of Egypt	Cairo	15
…	…	…	…

Indexes

- Note: When you drop a table, corresponding indexes are also dropped.

COUNTRY_ID	COUNTRY_NAME	CAPITOL	REGION_ID
1	United States of America	Washington, DC	21
2	Canada	Ottawa	21
3	Republic of Kazakhstan	Astana	143
7	Russian Federation	Moscow	151
12	Coral Sea Islands Territory	-	9
13	Cook Islands	Avarua	9
15	Europa Island	-	18
20	Arab Republic of Egypt	Cairo	15
…	…	…	…

Types of Indexes

- Two types of indexes can be created:
  – Unique index: The Oracle Server automatically creates this index when you define a column in a table to have a PRIMARY KEY or a UNIQUE KEY constraint.
    - The name of the index is the name given to the constraint.
    - Although you can manually create a unique index, it is recommended that you create a unique constraint in the table, which implicitly creates a unique index.
  – Nonunique index: This is an index that a user can create to speed up access to the rows.
    - For example, to optimize joins, you can create an index on the FOREIGN KEY column, which speeds up the search to match rows to the PRIMARY KEY column.

Creating an Index

- Create an index on one or more columns by issuing the CREATE INDEX statement:

```
CREATE INDEX index_name
ON  table_name( column...,column)
```

- To create an index in your schema, you must have the CREATE TABLE privilege.
- To create an index in any schema, you need the CREATE ANY INDEX privilege or the CREATE TABLE privilege on the table on which you are creating the index. Null values are not included in the index.

When to Create an Index

- An index should be created only if:
  - The 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 2-4% of the rows.

When Not to Create an Index

- When deciding whether or not to create an index, more is not always better.
- Each DML operation (INSERT, UPDATE, DELETE) that is performed on a table with indexes means that the indexes must be updated.
- The more indexes you have associated with a table, the more effort it takes to update all the indexes after the DML operation.

When Not to Create an Index

- It is usually not worth creating an index if:
  - The table is small
  - The columns are not often used as a condition in the query
  - Most queries are expected to retrieve more than 2-4 % of the rows in the table
  - The table is updated frequently
  - The indexed columns are referenced as part of an expression

Composite Index

- A composite index (also called a "concatenated" index) is an index that you create on multiple columns in a table.
- Columns in a composite index can appear in any order and need not be adjacent in the table.
- Composite indexes can speed retrieval of data for SELECT statements in which the WHERE clause references all or the leading portion of the columns in the composite index.

```
CREATE INDEX emps_name_idx
ON employees(first_name, last_name);
```

Confirming Indexes

- Confirm the existence of indexes from the USER_INDEXES data dictionary view.
- You can also check the columns involved in an index by querying the USER_IND_COLUMNS view.
- The query shown on the next slide is a join between the USER_INDEXES table (names of the indexes and their uniqueness) and the USER_IND_COLUMNS (names of the indexes, table names, and column names) table.

Confirming Indexes

```
SELECT DISTINCT ic.index_name, ic.column_name,
                ic.column_position, id.uniqueness
FROM user_indexes id, user_ind_columns ic
WHERE id.table_name = ic.table_name
AND ic.table_name = 'EMPLOYEES';
```

INDEX NAME	COLUMN NAME	COL POS	UNIQUENESS
EMP_EMAIL_UK	EMAIL	1	UNIQUE
EMP_EMP_ID_PK	EMPLOYEE_ID	1	UNIQUE
EMP_DEPARTMENT_IX	DEPARTMENT_ID	1	NONUNIQUE
EMP_JOB_IX	JOB_ID	1	NONUNIQUE
EMP_MANAGER_IX	MANAGER_ID	1	NONUNIQUE
EMP_NAME_IX	LAST_NAME	1	NONUNIQUE
EMP_NAME_IX	FIRST_NAME	2	NONUNIQUE

Function-based Indexes

- A function-based index stores the indexed values and uses the index based on a SELECT statement to retrieve the data.
- A function-based index is an index based on expressions.
- The index expression is built from table columns, constants, SQL functions, and user-defined functions.

Function-based Indexes

- Function-based indexes are useful when you don't know in what case the data was stored in the database.
- For example, you can create a function-based index that can be used with a SELECT statement using UPPER in the WHERE clause.
- The index will be used in this search.

```
CREATE INDEX upper_last_name_idx
ON employees (UPPER(last_name));
```

```
SELECT *
FROM employees
WHERE UPPER(last_name) = 'KING';
```

Function-based Indexes

- Another example of Function Based Indexes is shown here.
- The employees table is queried to find any employees hired in 1987.

```
SELECT first_name, last_name, hire_date
FROM employees
WHERE TO_CHAR(hire_date, 'yyyy') = '1987'
```

- This query results in a Full Table Scan, which can be a very expensive operation if the table is big.
- Even if the hire_date column is indexed, the index is not used due to the TO_CHAR expression.

Function-based Indexes

- Once we create the following Function Based Index, we can run the same query, but this time avoid the expensive Full Table Scan.

```
CREATE INDEX emp_hire_year_idx
ON employees (TO_CHAR(hire_date, 'yyyy'));
```

```
SELECT first_name, last_name, hire_date
FROM employees
WHERE TO_CHAR(hire_date, 'yyyy') = '1987'
```

FIRST_NAME	LAST_NAME	HIRE_DATE
Steven	King	17-Jun-1987
Jennifer	Whalen	17-Sep-1987

- Now, Oracle can use the index on the hire_date column.

Removing an Index

- You cannot modify indexes.
- To change an index, you must drop it and then re-create it.
- Remove an index definition from the data dictionary by issuing the DROP INDEX statement.

Removing an Index

- To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.
- If you drop a table, indexes and constraints are automatically dropped, but views and sequences remain.

```
DROP INDEX upper_last_name_idx;
```

```
DROP INDEX emps_name_idx;
```

```
DROP INDEX emp_hire_year_idx;
```

SYNONYM

- In SQL, as in language, a synonym is a word or expression that is an accepted substitute for another word.
- Synonyms are used to simplify access to objects by creating another name for the object.
- Synonyms can make referring to a table owned by another user easier and shorten lengthy object names.
- For example, to refer to the amy_copy_employees table in your classmate's schema, you can prefix the table name with the name of the user who created it followed by a period and then the table name, as in USMA_SBHS_SQL01_S04. amy_copy_employees.

SYNONYM

- Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence, procedure, or other object.
- This method can be especially useful with lengthy object names, such as views.
- The database administrator can create a public synonym accessible to all users and can specifically grant the CREATE PUBLIC SYNONYM privilege to any user, and that user can create public synonyms.

SYNONYM

- Syntax:

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

- PUBLIC: creates a synonym accessible to all users
- synonym: is the name of the synonym to be created
- object: identifies the object for which the synonym is created

```
CREATE SYNONYM amy_emps
FOR amy_copy_employees;
```

SYNONYM Guidelines

- Guidelines:
  - The object cannot be contained in a package.
  - A private synonym name must be distinct from all other objects owned by the same user.
- To remove a synonym:

```
DROP [PUBLIC] SYNONYM name_of_synonym
```

```
DROP SYNONYM amy_emps;
```

Confirming a SYNONYM

- The existence of synonyms can be confirmed by querying the USER_SYNONYMS data dictionary view.

Column Name	Contents
Synonym_name	Name of the synonym.
Table_name	Owner of the object referenced by the synonym.
Table_owner	Name of the object referenced by the synonym.
Db_link	Database link referenced in a remote synonym.