Working with Oracle SQL

Chapter 7:
Data Manipulation Language

Chapter Objectives

In this chapter, we will discuss:

- Manipulating data using DML:
  - `INSERT`
  - `UPDATE`
  - `DELETE`
- Transactional Control Statements:
  - `COMMIT`
  - `ROLLBACK`
  - `SAVEPOINT`

Chapter Concepts

INSERT

UPDATE

DELETE

Transactional Control

Chapter Summary

Data Manipulation Language Statements

- Data Manipulation Language (DML) is a classification of SQL
  - Others we are talking about:
    - Data Query Language (DQL)
      - The `SELECT` statement
    - Data Definition Language (DDL)
      - `CREATE`, `ALTER`
    - Transaction Control Language (TCL)
      - `COMMIT`, `ROLLBACK`, `SAVEPOINT`
- DML statements control the values of the data

Adding Data: the `INSERT` Statement

- DML statements allow you to manipulate the data in any table in your account
  - Or to have permission, if the table is owned by another user
    - If synonyms are not defined, qualify the table name: `schema.table`
- `INSERT`, `UPDATE`, and `DELETE` statements operate against a set of data
  - A set is zero, 1, or many rows
- The syntax for the `INSERT` allows either the insertion of one row or an entire set of rows

INSERTing One Row of Data

- Syntax:

```
INSERT INTO table_name [ (column_list) ]
VALUES (value_clause)
```

- Rules:
  - The values list must map to the column list one for one
  - All integrity rules must be met
  - If there is a defined default value for a column, it is invoked using the key word: `DEFAULT`
    - This table setting is covered later
  - The column list is optional
    - Values list must include a value for each column in the table in the order they appear in the table
      - Even if the value is `NULL`
      - Only use this for ad hoc inserts; always use the column list in production code

INSERTing One Row of Data: Examples

- The following are equivalent:

```
INSERT INTO regions
  VALUES (5, 'Antarctica');
```

```
INSERT INTO regions (region_id, region_name)
  VALUES (5, 'Antarctica');
```

```
INSERT INTO regions (region_name, region_id)
  VALUES ('Antarctica', 5);
```

```
INSERT INTO regions
  VALUES (5/1 ,SUBSTR('Antarctica', 1));
```

INSERTing One Row of Data: More Examples

- Unless a column is mandatory, a value does not have to be inserted
  - Columns for which no value is specified are set to NULL
  - Or to their DEFAULT value

```
INSERT INTO regions (region_id) VALUES (5);

1 row created.
```

```
SELECT *
FROM regions
WHERE region_id = 5;

 REGION_ID REGION_NAME
---------- -------------------------
         5
```

Accepting Default Values for Columns

- IF the `region_name` column had a `DEFAULT` value, then it would be used if a value was not inserted
  - Specify the key word `DEFAULT` if the column is in the column list

```
INSERT INTO regions (region_id) VALUES (5);
```

```
INSERT INTO regions (region_id, region_name) VALUES (5, DEFAULT);
```

  - The result in both cases would be:

```
 REGION_ID  REGION_NAME
----------  -------------------------
         5  Anytown
```

INSERTing a Set of Rows

- Syntax:

```
INSERT INTO table_name [ (column_list) ]
SELECT_clause
```

- Rules:
  - The SELECT clause can be any legal SQL statement
  - The SELECT list must map to the column list
  - The SELECT clause can return 0, 1, or more rows of data
  - The other rules are the same as for a single row INSERT
  - Each INSERT statement stands on its own
    - A data integrity violation with 1 row in the set causes the failure of all rows attempting to be inserted

INSERTing a Set of Data: Example

- Add rows from the location table into countries
  - Build the column being used as the primary key using the SUBSTR function

```
INSERT INTO countries (country_id, country_name)
SELECT SUBSTR(location_id, 1, 2), state_province
FROM locations;

23 rows created.
```

Chapter Concepts

INSERT

UPDATE

DELETE

Transactional Control

Chapter Summary

Maintaining Column Values: The `UPDATE` Statement

- The `UPDATE` modifies existing rows in the table
  - Changes the values of column(s) in 0, 1, or more rows in a table
- The columns can be set to static expressions
  - Including literals and functions
- Or to the returned values from a `SELECT` statement

UPDATE Syntax: Static Expressions

- Syntax:

```
UPDATE table_name
SET  column_name = expression | DEFAULT
          [  , … ]
[ WHERE condition ]
```

- Rules:
  - The key word *SET* is mandatory and appears only once
  - More than one column can be updated
    - The expression must resolve to the same datatype
  - The DEFAULT for the column can be referenced
  - The WHERE clause is optional
    - **If not specified, ALL rows of the table will be updated**

UPDATE Statement: Static Examples

```
UPDATE regions
SET region_name = 'NewTown'
WHERE region_id = 0;
```

```
SELECT *
FROM regions
WHERE region_id = 0;

 REGION_ID  REGION_NAME
----------  --------------------
         0  NewTown
```

```
UPDATE regions
SET region_id = 10 - 9
  , region_name = 'SubTown'
WHERE region_id = 0;
```

```
 REGION_ID  REGION_NAME
----------  ------------
         1  SubTown
```

DEFAULTs and NULLs

- The `DEFAULT` keyword can be used if one exists
- Be careful with `NULL`
  - `NULL` is a value
  - `'NULL'` is a literal string

```
UPDATE regions
SET    region_name = NULL
WHERE region_id = 0;
```

```
 REGION_ID REGION_NAME
---------- ----------------------
         0
```

```
UPDATE regions
SET    region_name = 'NULL'
WHERE region_id = 0;
```

```
 REGION_ID REGION_NAME
---------- ----------------------
         0 NULL
```

```
UPDATE regions
SET    region_name = DEFAULT
WHERE region_id = 0;
```

```
 REGION_ID REGION_NAME
---------- ----------------------
         0 Anytown
```

UPDATEing Using Subqueries

- A column can also be set equal to a subquery
- Syntax:

```
UPDATE table_name
SET   { (column [, column ]...) = (subquery)
        | [ column = { expr | (subquery) | DEFAULT }
      }
        [ , ... ]
WHERE conditional_clause
```

- Example:

```
UPDATE regions
SET region_name = (
    SELECT country_name
    FROM countries
    WHERE country_id = 'AR'
    )
WHERE region_id = 0;
```

```
 REGION_ID REGION_NAME
---------- ----------------------
         0 Argentina
```

UPDATE Syntax: Rules for Using Subqueries

- The subquery must be placed within parenthesis
- Specify a subquery that returns exactly one row for each row updated
  - If you specify only one column in the *update_set_clause*
    - The subquery can return only one value
- If you specify multiple columns in the *update_set_clause*:
  - The subquery must return as many values as you have specified columns
  - The multiple columns must be placed within parentheses
- If the subquery returns no rows, then the column is assigned a null

```
UPDATE regions
SET (region_id, region_name) = (
     SELECT 10, country_name
     FROM countries
     WHERE country_id = 'AR'
     )
WHERE region_id = 0;
```

Chapter Concepts

INSERT

UPDATE

DELETE

Transactional Control

Chapter Summary

Removing Row(s): The `DELETE` Statement

- Syntax:

```
DELETE FROM table
[WHERE conditional_clause]
```

  - The conditional clause specifies which rows are to be removed
  - The `WHERE` clause is optional
    - **If not specified, ALL rows of the table will be deleted**

DELETE Examples

- DELETE is a set statement

```
DELETE FROM emp;

15 rows deleted.
```

```
DELETE FROM emp WHERE sal > 6000;

1 row deleted.
```

```
DELETE FROM emp WHERE sal > 1500;

8 rows deleted.
```

```
DELETE FROM emp WHERE sal > 10000;

0 rows deleted.
```

DML and Integrity Constraints

- DELETE, like any DML statement, must conform to integrity constraints
  - Parent rows cannot be removed unless the CASCADE option is set

```
DELETE FROM regions WHERE region_id = 12;

1 row deleted.
```

```
DELETE FROM regions;

ERROR at line 1:
ORA-02292: integrity constraint (HR.COUNTR_REG_FK)
violated - child record found
```

Chapter Concepts

INSERT

UPDATE

DELETE

Transactional Control

Chapter Summary

Transactional Control

- A transaction is a logical unit of work that comprises one or more SQL statements
  - Must leave the database in a consistent state
- Data Manipulation Language (DML) statements are part of transactional control
  - The ability to make one or more changes to data as a group
  - The first SQL statement automatically starts a transaction
- To make the changes permanent to the database, issue a `COMMIT`
- To undo the changes, issue a `ROLLBACK`
  - `ROLLBACK` will undo all changes to the beginning of the transaction

Undoing Part of a Transaction

- `SAVEPOINT`s can be issued as part of the transactional stream
  - A savepoint is an intermediate marker that divides the transaction into smaller pieces
- Syntax:

```
SAVEPOINT my_savepoint;
```

- Allows the session to roll back any DML that was issued after the establishment of the savepoint

ROLLBACK Example

- First, set up the transactional activity in the session

```
INSERT INTO jobs VALUES ('IT_AC_PROG', 'Accounting Programmer', 1000, 1000);
1 row created.

SAVEPOINT sp_jobs;
Savepoint created.

INSERT INTO jobs VALUES ('IT_PR_PROG', 'Payroll Programmer', 999999, 999999);
1 row created.

SELECT job_id, job_title FROM jobs WHERE job_id IN ('IT_AC_PROG', 'IT_PR_PROG');
JOB_ID     JOB_TITLE
---------- -----------------------------------
IT_AC_PROG Accounting Programmer
IT_PR_PROG Payroll Programmer
```

ROLLBACK Example (continued)

- Undo the second insert by issuing the ROLLBACK TO... statement

```
ROLLBACK TO sp_jobs;
Rollback complete.
```

- Test for the existence of the inserts

```
JOB_ID     JOB_TITLE
---------- ----------------------------------
IT_AC_PROG Accounting Programmer
```

  - Note that the Payroll programmer no longer exists

- Finally, issue a session-level rollback and test again

```
ROLLBACK;
Rollback complete.

SELECT job_id, job_title FROM jobs WHERE job_id IN ('IT_AC_PROG', 'IT_PR_PROG');
no rows selected
```

- Where did we roll back to?

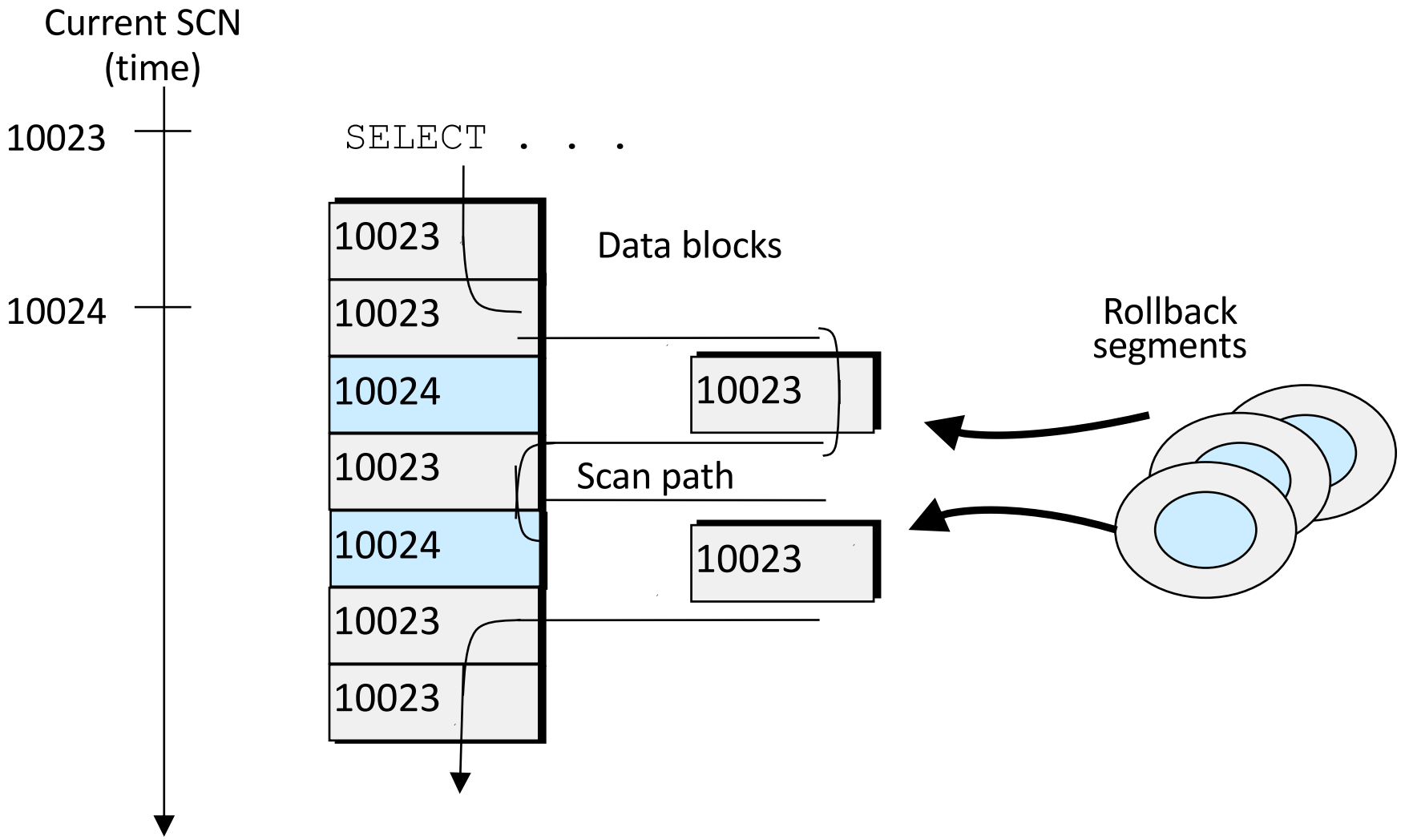
Read Consistency

- Statement-level read consistency
  - A single select is not impacted by database changes during its execution
- Transaction-level read consistency
  - Multiple selects in one transaction are not impacted by database changes
  - Enforced by `SET TRANSACTION READ ONLY`
- Ensures that readers of database data do not have to wait for writers or readers of the same data
- Ensures that writers of database data do not have to wait for readers of the same data
- Automatically enforced by Oracle using rollback segments and the System Change Number (SCN)

Read Consistency Example

- As query enters execution stage, the current SCN is determined
  - SCN is a sequential number assigned to each transaction
    - Recorded in rollback segment and redo log
  - In this example, SCN is 10023
- As query reads, only data blocks with SCN 10023 are used
- Blocks with changed data (more recent SCNs) are reconstructed using the rollback segments
- Reconstructed data is returned to the query
- Query returns all committed data with respect to the SCN recorded at the time execution started
- Committed or uncommitted changes of other transactions that occur during a query's execution are not seen, guaranteeing that a consistent set of data is returned for each query

Read Consistency Example (continued)



`SET TRANSACTION READ ONLY` Command

- First statement of a read-only transaction
- Remaining statements in the transaction cannot change data in the database
  - Insert, update, and delete are not allowed
- Last statement in the read-only transaction is `COMMIT` or `ROLLBACK`

Example of Transaction-Level Read Consistency

- Provide two listings of the `employees` table ordered by:
  - Last name
  - Hire date
- Transaction begins
  - Establish read consistency for the transaction
    ```
    SET TRANSACTION READ ONLY;
    ```
  - Create the employee listing by last name
    ```
    SELECT      *
    FROM  employees
    ORDER BY last_name;
    ```
  - Create the employee listing by hire date
    ```
    SELECT      *
    FROM  employees
    ORDER BY hire_date;
    ```
  - Release read consistency
    ```
    ROLLBACK;
    ```

Advantages and Disadvantages of Read Consistency

- Advantage:
  - Allows consistency of data throughout a transaction
- Disadvantages:
  - Performance of `SELECT` statements is impacted due to the necessity of reconstructing data from `ROLLBACK` segments
  - May get `snapshot too old` error

transaction isolation levels
- d**irty reads** - when transaction reads data that has not been committed
- norepeatable reads- read occur when a transaction reads a row twice but gets different data easch time
-phantoms - a row that matches search criteria but is not initially seen

read committed
read uncommitted
repeatable
serializable

Serializable Transactions

- Makes it appear as if there are no other users modifying data in the database
  - Any row read in the transaction is assured to be the same upon a reread
  - Most restrictive form of transaction isolation
- Set at the transaction level with the command
  `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE`
  - Causes all statements in the transaction, including `SELECT`, to obtain locks
- Improvement over `SET TRANSACTION READ ONLY` because it allows `UPDATE`, `INSERT`, and `DELETE`
- *Beware*—high cost to performance

Exercise 7.1: Manipulating Data

60 min

- Please complete this exercise in your Exercise Manual

Chapter Concepts

`INSERT`

`UPDATE`

`DELETE`

Transactional Control

Chapter Summary

Chapter Summary

In this chapter, we have discussed:

- Manipulating data using DML:
  - `INSERT`
  - `UPDATE`
  - `DELETE`
- Transactional Control Statements:
  - `COMMIT`
  - `ROLLBACK`
  - `SAVEPOINT`