Chapter 3: Oracle SQL - Data Manipulation and Transaction Control

Table of Contents

TRUNCATE TABLE Statement

When to Use TRUNCATE TABLE Instead of DELETE

DELETE is good for removing specific rows but:

- It's slower on large datasets
- It processes rows individually (updates indexes, fires triggers row-by-row)

TRUNCATE is more efficient for removing **all rows** in a table.

TRUNCATE TABLE Key Behaviors

- **Removes all rows** in the table
- **Removes all data** in the associated indexes
- **Fires no DML triggers**
- **Leaves table structure and index definitions intact**
- **Leaves dependencies intact** (like child tables)
- **Does not use undo space** like DELETE
- **Performs an implicit commit** (cannot be rolled back)
- **Cannot be used with FLASHBACK_TABLE**
- **Classified as DDL**, not DML

🔑 **Required Privilege**: DROP_ANY_TABLE

Why TRUNCATE Doesn't Fire Triggers

Since it's DDL, it **doesn't activate ON DELETE triggers**.

Syntax

```sql
TRUNCATE TABLE table_name;
```

Example:

```sql
TRUNCATE TABLE VENDORS;
```

Truncating with Child Tables (Oracle 12c+)

Use CASCADE when foreign key relationships exist with ON DELETE CASCADE:

```sql
TRUNCATE TABLE VENDORS CASCADE;
```

- TRUNCATE ... CASCADE works like DELETE ... ON DELETE CASCADE, but must be explicitly stated
- Without CASCADE, Oracle throws error ORA-02266 if dependent rows exist
- DELETE works without this clause because ON DELETE CASCADE handles the child rows automatically

Summary Comparison: DELETE vs TRUNCATE

Feature	DELETE	TRUNCATE
Row-by-row processing	✅ Yes	❌ No
Fires triggers	✅ Yes	❌ No
Uses undo space	✅ Yes	❌ No
Rollback possible	✅ Yes	❌ No
Implicit commit	❌ No	✅ Yes
Flashback table support	✅ Yes	❌ No
Privileges needed	DELETE privilege	DROP_ANY_TABLE
Type	DML	DDL

INSERT Statement

INSERT Statement Basics

- **INSERT INTO table_name**: Adds rows to a table
- Can insert **single** or **multiple rows**
- Works with **TABLES** and **certain VIEWs** (VIEWs must be updatable)

INSERT Syntax Breakdown

```sql
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

- Columns listed in parentheses (optional if inserting into all columns in order)
- Values must match columns **in number and compatible data types**
- You can omit columns with default values or NULL-allowed fields

Default Column List vs Explicit Column List

- You **can omit the column list**, but it's **risky**:
  - Table structure changes may break your code
  - Column order confusion (especially when types are similar)
- **Best practice**: **Always specify column names**

Valid Variations

You can:

- List columns in **any order** (just match value order)
- Omit non-required columns (e.g. with DEFAULT or NULL)

Data Type Conversion

- Oracle allows **implicit conversion** (e.g., '101' into a numeric column)
- Conversion must be **compatible**, not identical
- Not foolproof -- better to use **explicit conversions** (TO_CHAR, TO_NUMBER, etc.)

INSERT and Constraints

- Violations (e.g., CHECK, NOT NULL) cause **run-time errors**, not syntax errors

- Example: If a CHECK constraint allows only 'Hawaii', 'Mexico', then 'Hawaii and Back' will fail at runtime
- Even if INSERT fails, **sequences still increment**

Using SEQUENCES

Use sequence to auto-generate primary key:

```sql
INSERT INTO table_name (id_column, name_column)
VALUES (sequence_name.NEXTVAL, 'Some Value');
```

Example Table: CRUISES

```sql
CREATE TABLE CRUISES (
    CRUISE_ID NUMBER,
    CRUISE_NAME VARCHAR2(30),
    START_DATE DATE,
    END_DATE DATE,
    CONSTRAINT CRUISE_ID_PK PRIMARY KEY (CRUISE_ID),
    CONSTRAINT CRUISE_NAME_CK CHECK (
        CRUISE_NAME IN ('Hawaii', 'Bahamas', 'Bermuda', 'Mexico', 'Day at Sea')
    )
);
```

⚠️ Pitfalls to Avoid

- Don't rely on **default column order**
- Always ensure **data types are compatible**
- Respect all **constraints** -- CHECK, NOT NULL, etc.
- Understand that **implicit conversion** can mask errors

UPDATE Statement

UPDATE Statement Essentials

- **Purpose**: Modifies existing data in a single table (or a view based on a table)
- **Syntax**:

```sql
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Key Elements of UPDATE

1. **UPDATE table_name** -- Specifies the table to be updated
2. **SET clause** -- Lists column(s) to update with new values:
   - Format: column_name = expression
   - Expression can be: literal value, column reference, math expression, or SQL function
   - Multiple assignments separated by commas
3. **WHERE clause** *(optional but critical)*:
   - Filters which rows to update
   - If omitted, **all rows** in the table will be updated

Important Behaviors & Gotchas

- **No particular order** is needed for columns listed in SET
- **Not required** to update all columns
- Columns omitted from SET remain unchanged
- **Cannot violate constraints** (e.g., NOT NULL, CHECK):
  - If *any* row violates a constraint, the **entire update fails**
- **UPDATE doesn't care** whether the current value is NULL or already has a value—it just overwrites

Expressions in SET Clause

Example:

```sql
UPDATE COMPENSATION
SET SALARY = SALARY * 1.03,
    LAST_CHANGED_DATE = SYSDATE
WHERE EMPLOYEE_NUMBER = 83;
```

- SALARY is increased by 3%
- SYSDATE sets the timestamp

Constraint Violation Example

```sql
-- Table has CHECK constraint: COST < 1,000,000
UPDATE PROJECTS
SET COST = COST * 1.20;
```

If any row violates the CHECK, **entire UPDATE fails**.

Fix:

```sql
UPDATE PROJECTS
SET COST = COST * 1.20
WHERE COST * 1.20 < 1000000;
```

Exam Watch & Practical Notes

- **Don't confuse** SET in UPDATE with SET operators like UNION, INTERSECT, or MINUS
- On the job, "adding data" might mean using UPDATE to set a NULL column—watch for language confusion
- You can "remove" data using UPDATE column = NULL, but **removing a row** requires DELETE

DELETE Statement

Syntax

```sql
DELETE FROM table_name WHERE condition;
-- or
DELETE table_name WHERE condition;
```

Key Points

- **Deletes entire rows**. To nullify a value within a row, use UPDATE, not DELETE
- If WHERE is omitted, **all rows** in the table are deleted

Transaction Control Language (TCL)

TCL statements manage transactions for DML statements (INSERT, UPDATE, DELETE).

COMMIT

Makes changes permanent.

- Cannot be undone once executed
- Two types:
  - **Explicit**: `COMMIT;` or `COMMIT WORK;`
  - **Implicit**: Happens automatically:
    - Before/after executing DDL (e.g. CREATE, ALTER, DROP)
    - On normal exit from tools like SQL*Plus or SQL Developer
- Changes are **not visible** to other sessions until a COMMIT occurs (explicit or implicit)

ROLLBACK

Undoes changes made in the session **since the last COMMIT**.

- Only affects **uncommitted changes**
- Syntax:

```sql
ROLLBACK;                    -- full rollback to last COMMIT
ROLLBACK TO savepoint_name;  -- partial rollback to a savepoint
```

- If the rollback references a **nonexistent savepoint**, it fails with an error, and the database remains unchanged

SAVEPOINT

Marks a point within a transaction for partial rollback:

```sql
SAVEPOINT sp_name;
ROLLBACK TO sp_name;
```

SAVEPOINTS are:

- **Named**
- **Volatile** — erased after COMMIT
- **Optional** for more granular control over rollbacks

Example:

```sql
COMMIT;
UPDATE ...;
SAVEPOINT mark1;
UPDATE ...;
ROLLBACK TO mark1;
COMMIT;
```

Savepoints allow **selective undo** of changes **before the next COMMIT**.

Session Behavior

- **Uncommitted changes** are visible **only within the same session**
- **Other sessions**, including those of the same user, **won't see changes** until COMMIT
- This includes both interactive tools and automated jobs/scripts

---

Quick Reference Summary

TRUNCATE TABLE

- Used to **quickly remove all rows** from a table
- **No DML triggers** fired, and **indexes remain**
- Considered **DDL**, not DML → **Implicit COMMIT** is performed
- **Cannot be rolled back**
- Syntax: `TRUNCATE TABLE table_name;`
- With foreign keys: `TRUNCATE TABLE table_name CASCADE;`

INSERT INTO

- Adds one or more rows
- Syntax: `INSERT INTO table_name [(col1, col2, ...)] VALUES (val1, val2, ...);`
- If **column list is omitted**, value order must match table's column structure
- Honors **constraints** (e.g., NOT NULL)
- Fails if data types or constraints are violated

UPDATE

- Modifies data in existing rows
- Syntax: `UPDATE table_name SET col1 = val1 [, col2 = val2, ...] WHERE condition;`
- If WHERE is omitted, **all rows** are updated
- Expressions allowed in value assignments (e.g., functions, calculations)

DELETE

- Removes rows from a table
- Syntax: `DELETE FROM table_name WHERE condition;`
- If WHERE is omitted, **all rows** are deleted

TCL Statements

- **COMMIT**: Makes changes permanent (explicit or implicit)
- **ROLLBACK**: Reverses uncommitted changes
- **SAVEPOINT**: Creates named point for partial rollbacks