Orange Coast College
Business Division - CS/CIS Department
CIS 183 – Oracle™ SQL Programming
Instructor: Dr. Martha Malaty

Text & Original Presentations: Joan Casteel
Oracle 11g: SQL
Chapter 5
Data Manipulation & Transaction Control
Agenda

- Solve chapter 3 assignment
- Part 1 of Lecture
- Break
- Part 2 of Lecture
- Hands-On assignment
Chapter Objectives

- Add a record to an existing table
- Add a record containing a NULL value to an existing table
- Use a subquery to copy records from an existing table
- Modify the existing rows within a table
- Use substitution variables with an **UPDATE** command
- Issue & manage transaction control statements **COMMIT** and **ROLLBACK**, and **SAVEPOINT**
- Differentiate between **DDL, DML**, and transaction control commands
- Delete records
- Differentiate between a shared lock and an exclusive lock and use the **SELECT...FOR UPDATE** command to create a shared lock
Commands Used

- INSERT
- UPDATE
- DELETE
- COMMIT
- ROLLBACK
- SAVEPOINT
- LOCK TABLE
- SELECT .. FOR UPDATE
- Substitution Variable & (Interactive operator)

Note:
- Before starting to work on this chapter, run the “JLDB_Bind_5.sql” script.
What is a Database Transaction?

- **Wikipedia definition:**
  - A unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions.
  - Transactions have two main purposes:
    1. Provide **reliable units of work** that allow correct recovery from failures and keep a **database consistent** even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.
    2. Provide **isolation** between programs accessing a database **concurrently**. If this isolation is not provided the programs outcome are possibly erroneous.
What is a Database Transaction?

- **Wikipedia definition** (Continued):
  - A database transaction, by definition, must have the **ACID** property
    - Atomic
    - Consistent
    - Isolated
    - Durable

- **SQL DML operations** are used in transactions
  - A transaction is a **collection of DML statements** that form a **logical unit of work**
  - Transaction **duration** is defined when a **COMMIT** command is issued
What is a Database Transaction?

- **Wikipedia definition** (Continued):
  - Must provide an "all-or-nothing" proposition
    - Each work-unit performed in a database must either
      - Complete in its entirety, or
      - Have no effect whatsoever
    - The system must isolate each transaction from other transactions
    - Results must conform to existing constraints in the database'
    - Transactions that complete successfully must get written to durable storage
DML statements:

- **INSERT**
  - Add new rows to a table
- **UPDATE**
  - Modify existing rows in a table
- **DELETE**
  - Delete rows from a table
You can control the logic of transactions by using the **COMMIT**, **SAVEPOINT**, and **ROLLBACK** statements.

Use **COMMIT** and **ROLLBACK** statements to:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations
A customer transfers money from account A to account B

The transaction consists of three separate operations
- Decrease the amount from account A
- Increase the account B with the same amount
- Record the transaction in a transaction log

The database system must guarantee that all three operations are performed to maintain the proper balance

If any of the statements has an error and cannot be executed, the other operations of the transaction must be canceled, even if they are correct
INSERT Command

- Used to **add new rows** to existing tables
- Identify table in the **INSERT INTO** clause
- Specify data in the **VALUES** clause
- Can only add **one row at a time** to a table
- Different column sequence can be chosen, in which case you should give the column list in the **INSERT INTO** clause

**Syntax:**

```
INSERT INTO table [(column [, column...])] VALUES (value [, value...]);
```
INSERT Command Examples

No Column List

```sql
INSERT INTO acctmanager
VALUES ('T500', 'NICK', 'TAYLOR', '05-SEP-09', 42000, 3500, 'NE');
```

1 rows inserted

Column List

```sql
INSERT INTO acctmanager (amid, amfirst, amlast, amsal, amcomm, region)
VALUES ('J500', 'Sammie', 'Jones', 39500, 2000, 'NW');
```

1 rows inserted
Numeric values:
- Do not enclose in single quotation marks
- Will accept quotation, but will have more overhead
  - Implicitly converted from character to numbers

Characters and dates:
- Enclose in single quotation marks
- Character data retain the case in which they were entered in the INSERT INTO statement

If column list is not provided, a value must be assigned to each column in the table, in the same order the columns were created

For clarity, use the column list in the INSERT clause
Ways to Insert NULL Value

1. Implicit:
   ◦ **Omit column name** from **INSERT INTO** clause column list

2. Explicit:
   ◦ **Include **NULL** keyword** in the positions in the **VALUES** clause
   ◦ For character or date types, substitute **two single quotation marks** in the position in the **VALUES** clause

![SQL Insert Statement Example](image-url)

- NULL value input

---

**SQL Statement Example:**

```sql
INSERT INTO acctmanager
VALUES ('L500', 'MANDY', 'LOPEZ', '01-0CT-09', 47000, 1500, NULL);
```

- 1 rows inserted
Inserting NULL Value

- Be sure that you can use **NULL** values in the targeted column by **checking the status of the column** using the **DESCRIBE** command
  - The **DBMS automatically enforces**
    - **All data types**
    - **Data ranges**, and
    - **Data integrity constraints**

- Any column that is not listed explicitly in the column list will have a null value in the new row
Manage Virtual Column Input

- Oracle 11g:
  - Remember that you cannot insert values into virtual columns

![Error message for inserting values into virtual columns](image-url)
Common Errors While Inserting

- When you add or modify table data, the data is checked for compliance with any applicable constraints
  - **Missing value** for a **NOT NULL** column
  - **Data type mismatch**
  - **Value too wide to fit in column**
  - Inserting value in a **virtual column**
  - **Constraint violation:**
    - **Duplicate value violates unique constraint**
    - **Foreign key constraint violated**
    - **CHECK constraint violated**
Inserting NULL Value Example

- Check acctamanager2 structure
  - amlast cannot be NULL
- Try to insert NULL value, will get error
Inserting NULL Value Example

- Now insert a value other than NULL

```
INSERT INTO acctmanager2
VALUES ('L500', 'Martha', 'Malaty', SYSDATE, 'NW');
```

1 row(s) inserted.
0.20 seconds
We can insert NULL value in amdate field.
Now ignore the date field

- Even when no date was entered, the date field of the last record has a value which is the default system date.
Constraint Violations

- When you **add** or **modify** table data, the data is checked for compliance with any applicable constraints
Activating the DEFAULT option

- Include a column list in the **INSERT** statement ignoring the column to use the **DEFAULT** option.
- Use the **DEFAULT** keyword as the value for the column.
Substitute subquery for **VALUES** clause

Syntax:

```sql
INSERT INTO tablename [(columnname, ...)]
subquery;
```

Notes

- The keywords "**VALUES**" and "**AS**" are not used
- The subquery is not enclosed in parentheses
- The number and data types of columns in the column list of the **INSERT** clause must match the number of values and data types in the subquery
Example:

```sql
INSERT INTO acctbonus (amid, amsal, region)
SELECT amid, amsal, region
FROM acctmanager;
```
Modifying Existing Rows

- Modify rows using **UPDATE** command
- Use **UPDATE** command to:
  - Add values to an existing row (replace NULL values)
  - Change existing values
- Syntax:

```sql
UPDATE table_name
SET column_name = new_value,
    ...
[WHERE condition];
```
Modifying Existing Rows

- You can **change more than one column** value in the same `UPDATE` command
- You can use **subqueries** to get the value of the column
- Syntax:

```sql
UPDATE tablename
SET columnname =
    (SELECT columnname
     FROM tablename
     WHERE condition)
    [, columnname =
        (SELECT columnname
         FROM tablename
         WHERE condition)]
[[WHERE condition ] ;
```
Modifying Existing Rows

- **UPDATE** clause identifies the table
- **SET** clause identifies the column being changed and new value
- Optional **WHERE** clause specifies row(s) to be changed – if omitted, will update all rows
- To Modify a single row, use the primary key to select that row
  - Using other columns can unexpectedly cause several rows to be updated
UPDATE Command Example

```sql
UPDATE acctmanager
SET amedate = '10-OCT-09',
    region = 'S'
WHERE amid = 'L500';
```

1 rows updated
Substitution Variables

- Used to create interactive scripts
- Substitutes the value of a variable at runtime
- Prompts the user for a value
- Identified by ampersand('&') followed by a variable name
- The script can be saved to a file and executed several times
  - We can supply different set of values each time it is run
- for command line, use a dash('-') character to continue a command on the next line
Substitution Variable Example

Update customers
Set region = '&Region' 
Where state = '&State';

Enter Substitution Variable

REGION:

W

OK Cancel
SET VERIFY ON/OFF

- Valid for Command line interface
- If you don't want to see the duplicate display of the old and new values, use `SET VERIFY OFF`
- If you want it back use `SET VERIFY ON`
You can use substitution variables in:
- Command Line
- Express edition
- Oracle Server

Check directions for each in Web Links sections
SET verify on/off example

- In Oracle Express 10g
  - Use : instead of the &
  - Remove the quotes

Example:

```sql
UPDATE customers
SET region = :Region
WHERE state = :State;
```

```
SQL> SET VERIFY OFF;
SQL> UPDATE customers
2  SET region = '&Region'
3  WHERE state = '&State';
Enter value for region: NE
Enter value for state: MA

1 row updated.
```

```
SQL> SET VERIFY ON;
SQL> UPDATE customers
2  SET region = '&Region'
3  WHERE state = '&State';
Enter value for region: NE
Enter value for state: MA

old   2: SET region = '&Region'
new   2: SET region = 'NE'
old   3: WHERE state = '&State'
new   3: WHERE state = 'MA'

1 row updated.
```
For Command line interface
Contains the last SQL command
If you type RUN or the forward slash '/' followed by the Enter key, the last command will be re-executed

SQL> /
Enter value for region: NE
old  2: SET region = '&Region'
new  2: SET region = 'NE'
Enter value for state: MA
old  3: WHERE state = '&State'
new  3: WHERE state = 'MA'

1 row updated.
Deleting Rows

- **DELETE** command removes entire row(s) from a table if this does not violate referential integrity constraints
  - If the table is the parent of a referential integrity constraint, you still can remove the row if you disable the constraint

- You can delete specific rows by specifying the **WHERE** clause in the **DELETE** command
  - If no **WHERE** clause is specified, all rows will be deleted

- Remember that the delete operation is not made permanent until the data transaction is committed

- Syntax:
  
```sql
DELETE FROM tablename
[WHERE condition];
```
Deleting Rows

- To **remove all the rows** from a table, you better use the **TRUNCATE** command.
- Removing rows with the **TRUNCATE** statement is **faster** than removing them with the **DELETE** statement because:
  - The **TRUNCATE** statement is a data definition language (DDL) statement and generates **no rollback** information.
  - Truncating a table **does not fire the delete triggers** of the table.
  - If integrity constraint is violated, the truncate command will fail.
Deleting Rows Example

WHERE clause determines which row(s) are removed

DELETE FROM acctmanager
WHERE amid = 'J500';

1 rows deleted
Caution!!!

- Omitting WHERE clause removes all rows
- You should uncheck the “Autocommit” checkbox in the Express edition, otherwise the deletion will be permanent
Results of data manipulation language (DML) are not permanently updated to a table until explicit or implicit COMMIT occurs.

Transaction control statements can:

- Commit data through COMMIT command
- Undo data changes through ROLLBACK command
COMMIT Command

- **Ends** the current transaction
  - Makes all pending data **changes permanent**
  - Allows other **users to view changes**

- **Types of COMMIT**
  - **Explicit:**
    - When you execute `COMMIT` command at the SQL prompt
  - **Implicit:**
    - At the end of the SQL session by typing `EXIT``
    - When DDL command are performed (e.g. `CREATE TABLE`, `ALTER TABLE`)
ROLLBACK Command

- Ends the current transaction by **discarding** all pending data changes
- Used to “**undo**” changes that have not been committed
- Occurs when:
  - **ROLLBACK**; is executed
  - **System restarts** after crash
- **SAVEPOINT** marks a specific spot within the transaction
- Can **ROLLBACK** to a **SAVEPOINT** to undo part of the transaction
Transaction Control Example

```sql
UPDATE acctmanager
SET region = 'E'
WHERE amid = 'M500';
COMMIT;

UPDATE acctmanager
SET region = 'E'
WHERE amid = 'T500';

UPDATE acctmanager
SET amcomm = 6600
WHERE amid = 'T500';

SAVEPOINT ONE;

UPDATE acctmanager
SET region = 'E'
WHERE amid = 'L500';

SAVEPOINT ONE succeeded.
1 rows updated
COMMIT succeeded.
1 rows updated
1 rows updated
SAVEPOINT ONE succeeded.
1 rows updated
```
TRANSACTION CONTROL EXAMPLE (Continued)

Enter SQL Statement:

ROLLBACK TO ONE;

Only undo DML actions after SAVEPOINT

ROLLBACK TO succeeded.
Save Points

- During a long transaction, that has **partial transactions**, you can **bookmark** some points in the transaction and rollback to those points if necessary

  - **SAVEPOINT** marks a specific spot within the transaction

  - Can **ROLLBACK** to a **SAVEPOINT** to undo part of the transaction

- Syntax:
  
  ```plaintext
  SAVEPOINT name;
  &
  ROLLBACK TO SAVEPOINT name;
  ```
Save Points

- **ROLLBACK TO SAVEPOINT** rolls back the current transaction to the specified save point.

- Any changes and/or save points created after the save point to which you are rolling back will be discarded.

- Omitting **TO SAVEPOINT** clause rolls back the entire transaction.

- There is no way to list the save points you have created, since they are only logical concepts.
What is Database Lock?

- Wikipedia definition
  - A lock is used when multiple users need to access a database concurrently
  - Prevents data from being corrupted or invalidated when multiple users try to write to the database
  - Any single user can only modify those database records to which they have applied a lock that gives them exclusive access to the record until the lock is released
  - Also prevents reading of unfinished modifications to uncommitted data
What is Database Lock?

- **Wikipedia definition (Continued)**
  - Two mechanisms for locking data in a database:
    1. **Pessimistic locking**
       - A record or page is locked *immediately when the lock is requested*
       - Is **guaranteed** that the record will be updated
    2. **Optimistic locking**
       - A record is only locked when the changes made to that record are updated
       - Only appropriate when there is less chance of someone needing to access the record while it is locked
Table Locks in Oracle

- Can be applied to
  - Tables
  - Rows
  - Data dictionary rows

- Performed automatically by Oracle

- Requires no user action

- Held for the duration of the transaction
Implicit locking:
- Oracle automatically locks the rows whenever user performs DML operations
- Occurs for all SQL statements except `SELECT`
- (e.g. during `UPDATE` or `DELETE` operations)

Explicit locking:
- Users can lock data manually
  - Through `LOCK TABLE` command with `SHARE MODE` option
Table Lock Levels

- **Row level:**
  - Lock selected rows of table
  - Imposed by "for update" clause in select statement

- **Table level:**
  - Lock complete table

- Locks are held until commit or rollback

- More in [Oracle Docs](#)
Locks Usage

- **DML:**
  - Table share
  - Row exclusive

- **Queries:**
  - No locks required

- **DDL:**
  - Protects object definitions
  - Occur when you modify a database object such as a table
Exclusive Lock

- Prevent the row from being changed by other transactions until the transaction is committed or rolled back
- Locks out other users or other exclusive or shared locks
- Released after execution of DDL operation or after user exits system
- Can be implicit or explicit
Shared Locks

- Locks the portion of the table affected by DML operation
- **Other users can view** the data in the table but **cannot alter** the structure or perform any DDL operations
- Several transactions can share the same resource
- Implicit
  - During **UPDATE** or **DELETE** operations
- Explicit
  - Through **LOCK TABLE** command with **SHARE MODE** option
- Released when **COMMIT** (implicit or explicit) or **ROLLBACK** occurs
SELECT...FOR UPDATE Command

- Creates shared lock on retrieved portion of table
- Prevents one user from changing a row while another user is selecting rows to be changed
- Released through implicit or explicit commit
SELECT...FOR UPDATE Command

```
SELECT columnnames,...
FROM tablename, ...
[WHERE condition]
FOR UPDATE;
```

**FIGURE 5-34**  Syntax for the SELECT...FOR UPDATE command

**Example**

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT *
2  FROM acctmanager
3  WHERE amid = 'L500'
4  FOR UPDATE;

AMID  AMNAME       AMEDATE   RE
------ -------------- --------- ----
L500   HANDY LOPEZ  01-OCT-02 SE

SQL> COMMIT;
Commit complete.
SQL>
```
Summary

- **INSERT INTO** adds new rows to existing table
- **NULL** value entered as **NULL**, omitting column from column list, or entering 2 single quotation marks
- No **VALUES** keyword used when inserting from one table into another
- **UPDATE** can update contents of a row
- Substitution variables are used to allow a command to be executed several times with different data values
- **COMMIT** commits the transactions performed by DML commands
- Uncommitted DML commands can be undone using **ROLLBACK** command
- **DELETE** command removes rows from a table depending on a condition
- In multi-user environment, table locks are used to prevent users from mistakenly overwriting changes made by other users