

Chapter 11

Insert, Update, Delete

Referencing Another User's Objects (nib)

- Use the owner's schema name as a prefix to the object name

```
SELECT *  
FROM ttrollen.writer;
```

- ... if you've been granted privileges to do so

2

Data Manipulation Language

- Use DML statements to
 - INSERT new rows to a table
 - UPDATE existing rows in a table
 - DELETE existing rows from a table

3

Inserting an Individual Row

```
INSERT INTO table [(column [, column]...)]  
VALUES (expression|DEFAULT)[, (expression|DEFAULT)]...;
```

- You can omit the field names if you provide a value for every field
 - values are positionally matched to the table's fields

```
INSERT INTO writer  
VALUES ('A123', 'Along', 'Bob', Null, Null, Null, Null, Null);
```

- You must specify field names if you don't provide a value for each field or specify the values in a different order

```
INSERT INTO writer(ln, fn, writerid)  
VALUES ('New', 'Writer', 'N999');
```

```
COMMIT;
```

4

Inserting Multiple Rows

- Can insert rows selected from existing table(s)
 - the field names from the two tables are positionally matched

```
INSERT INTO student
(STUDENT_ID, LAST_NAME, FIRST_NAME, ZIP,
REGISTRATION_DATE, CREATED_BY, CREATED_DATE, MODIFIED_BY, MODIFIED_DATE)
SELECT student_id_seq.NEXTVAL, LAST_NAME, FIRST_NAME, ZIP,
SYSDATE, USER, SYSDATE, USER, SYSDATE
FROM instructor
WHERE zip IS NOT NULL;
```

Statement Output x

9 rows inserted

```
ROLLBACK;
```

5

Transaction Control

- A **transaction** consists of a collection of DML statements that form a logical unit of work
 - used to group a series of changes into a single **batch**
 - help assure **data integrity**
 - classic example: transferring money between accounts (next slide)
 - an **all or nothing** proposition
 - if each and every operation in the batch succeeds, use the **COMMIT** command write the data to the database
 - if any of the individual operations in the batch fail, use **ROLLBACK** command to undo pending (uncommitted) DML statements
 - can create an intermediate marker in a current transaction by using the **SAVEPOINT** statement
 - can rollback to an intermediate marker using the **ROLLBACK TO SAVEPOINT** statement

6

Transaction Control: Example

- An ATM user transfers \$500 from her savings account to her checking account.

```
UPDATE account SET balance = balance - 500
WHERE account_number = 44533277658;

UPDATE account SET balance = balance + 500
WHERE account_number = 44533277650;

INSERT INTO transfer_log
VALUES (logseq.NEXTVAL, 44533277658, 44533277650, 500, SYSDATE);

COMMIT;
```

7

Transaction

- Begins when the first DML statement is executed
- Ends with one of the following events:
 - COMMIT** is explicitly issued
 - ROLLBACK** is explicitly issued
 - DDL statement executes (automatic COMMIT)
 - egs: CREATE, ALTER, DROP (Ch12)
 - DCL statement executes (automatic COMMIT)
 - egs: GRANT, REVOKE (Ch15)
 - User exits SQL*Plus/SQL Developer normally (automatic COMMIT)
 - Abnormal termination of SQL*Plus /SQL Developer (automatic ROLLBACK)
 - Close button
 - Task Manager to end task
 - Oracle server crashes (automatic ROLLBACK)

8

State of the Data During a Transaction (Before a COMMIT or ROLLBACK)

- The previous state of the data can be recovered
 - rollback segments have the old values
- The user can review the results of the session's own DML operations by using a SELECT statement
- Other sessions **cannot** view the results of the DML statements by the current session (**read consistency**)
- The affected rows are **locked**
 - other sessions cannot change the data within the affected rows

9

Practice Time

- Multi-User Environment
 - what is happening and why

```
SELECT * FROM ttrollen.writer
WHERE WRITERID = 'J525';

UPDATE ttrollen.writer
SET fn = 'Tom'
WHERE WRITERID = 'J525';

SELECT * FROM ttrollen.writer
WHERE WRITERID = 'J525';

COMMIT;
```

- in the end, what fn value is stored for writer 'J525'

10

State of the Data After a Transaction

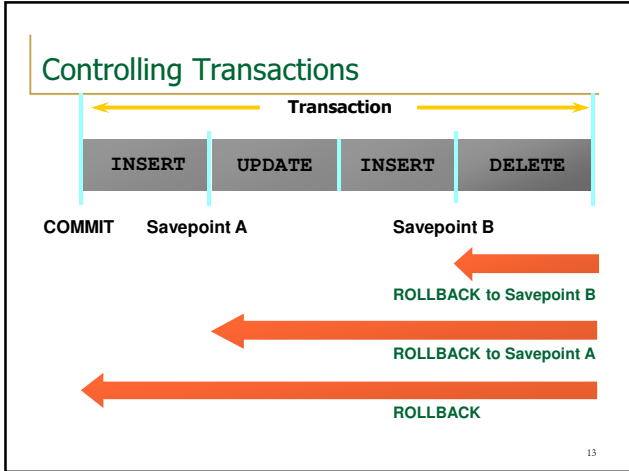
- After a **COMMIT**
 - Data changes are written to disk
 - Locks on the affected rows are released
 - Other sessions can now view the results
- After a **ROLLBACK**
 - Pending data changes are undone
 - The previous state of the data is restored from the rollback segments
 - Locks on the affected rows are released
 - Statement-Level Rollback (p445)
 - A misnomer
 - "If one individual statement fails in a series of DML statements, only this statement is rolled back"
 - All preceding Insert/Update/Delete changes are retained in the pending, uncommitted transaction

11

Read Consistency

- After User A modifies data, he can use SELECT statements to see his session's changes
- User B will see the previous state of the data until User A commits his session's pending transaction
 - User B does not see the User A's **uncommitted changes**
- Oracle stores the old values in the Undo Tablespace
 - old values are available in case the transaction is rolled back
 - old values are used to present other users a read consistent view of data
 - previous versions of Oracle called them Rollback Segments

12



Using Special Characters in SQL Statements (p450)

- In SQL*Plus and SQL Developer, **&** indicates that what follows is a **substitution variable** (Ch14)
- SET DEFINE [on|off]
 - controls whether or not **&** signals a substitution variable
- SET VERIFY [on|off]
 - see/suppress substitution variable details

The screenshot shows a query window with the following SQL: `SELECT title, length FROM article WHERE title = 'AT&T Antitrust Settlement';`. An 'Enter Substitution Variable' dialog box is open, showing 'T:' and the value 'eee'. Below the dialog, the SQL is shown with 'SET DEFINE OFF' and 'SET DEFINE ON' options. The query result shows one row: 'AT&T Antitrust Settlement' with a length of 1600.

Using Special Characters in SQL Statements (p451)

- Text literals must be enclosed in single quotes
 - 'Burger King'
- What if the string data contains a single quote
 - 'Wendy's'
 - use a pair of single quotes
 - 'Wendy's'

UPDATING Rows

Lab 11.2

```
UPDATE table
SET col={expr|DEFAULT|NULL}[, col={expr|DEFAULT|Null}]...
[WHERE condition];
```

- All rows in the table will be modified if you omit the WHERE clause!
- Only specific row(s) are modified when you specify a WHERE clause

```
UPDATE writer
SET amount = amount + 200, lastcontact = SYSDATE
WHERE writerid = 'J525';

ROLLBACK;
```

UPDATING Rows and CASE

- Can use a searched CASE expression to specify conditional logic that is evaluated before column value is updated

```
UPDATE writer
  SET amount = CASE WHEN contact IS NOT NULL
                    THEN amount + 200
                    WHEN contact IS NULL
                    THEN amount + 500
                END
WHERE lastcontact > (SYSDATE - 30);
```

17

DELETING Rows

```
DELETE [FROM] table
[WHERE condition];
```

- All rows in the table will be deleted if you omit the WHERE clause!

```
DELETE FROM writer WHERE writerid in ('A123','N999');
COMMIT;
```

- If the table is the One side of a One-to-Many relationship and the row(s) to be deleted have matching rows on the Many side
 - if the foreign key constraint uses the ON DELETE CASCADE option the related rows in the Many side of the relationship will be deleted without warning
 - if the foreign key constraint does not use the ON DELETE CASCADE option you won't be able to delete the requested row(s)
 - ORA-02292 error

```
DELETE writer WHERE writerid = 'J525';
ROLLBACK;
```

18

TRUNCATE Statement

```
TRUNCATE TABLE tablename
```

- Deletes all rows from the specified table and immediately issues an automatic COMMIT
 - can't use ROLLBACK!... be careful!
 - any preceding DML changes committed as well!
- What are you left with?
- Does not allow a WHERE clause

```
TRUNCATE TABLE junk
```

19

Locks

- Oracle sets and releases **row-level locks** as transactions start/commit/rollback
 - when a row is locked by one session and then a second session attempts a DML operation, the second session cannot obtain a lock so it waits for the existing lock to be released
- Oracle sets and releases **table-level locks** during DDL operations (Chapters 12, 13)
 - egs: ALTER TABLE, CREATE INDEX

20

FOR UPDATE Clause (pg 472)

```
SELECT ... FROM ...
FOR UPDATE [{WAIT n | NOWAIT}]
```

- Locks the SELECTed rows so no other session can lock or update or delete them until you end your transaction
 - also prevents other sessions who also use SELECT... FOR UPDATE from seeing the existing values
 - "pessimistic locking"
- NOWAIT/WAIT clause tells Oracle how to proceed if the SELECT attempts to lock a row that is already locked
 - specify **NOWAIT** to return control to you immediately if a lock already exists
 - specify **WAIT** to instruct the database to wait **n** seconds for the row to become available and then return control to you

21

Lost Update (pg 471)

- What qty_on_hand is stored?
- What qty_on_hand should be stored?

- Occurs when one session's update is overwritten by another's

Session 1 (sell 3 units)	Session 2 (sell 1 unit)
<pre>SELECT qty_on_hand FROM inventory WHERE partnum=4552; QTY_ON_HAND ----- 307</pre>	
<pre>UPDATE inventory SET qty_on_hand=304 WHERE partnum=4552;</pre>	
	<pre>SELECT qty_on_hand FROM inventory WHERE partnum=4552; QTY_ON_HAND ----- 307</pre>
<pre>COMMIT;</pre>	<pre>UPDATE inventory SET qty_on_hand = 306 WHERE partnum = 4552;</pre>
	<pre>COMMIT;</pre>

Lost Update (pg 471)

- 3 Solutions
 - Include a safeguard in the UPDATE statement's WHERE clause

```
UPDATE inventory
SET qty_on_hand = 306
WHERE partnum = 4552 AND qty_on_hand =307;
```

No rows updated.

- Have the UPDATE statement calculate the value dynamically

```
UPDATE inventory
SET qty_on_hand = qty_on_hand - 1
WHERE partnum = 4552;
```

1 row updated.

- Use the FOR UPDATE clause when performing the SELECT

```
SELECT qty_on_hand
FROM inventory
WHERE partnum = 4552 FOR UPDATE NOWAIT;
```

T4

T4

T1

23

Using SELECT... FOR UPDATE to Prevent a Lost Update

<pre>SELECT qty_on_hand FROM inventory WHERE partnum=4552 FOR UPDATE NOWAIT; QTY_ON_HAND ----- 307</pre>	<p>Second user can't obtain lock, cannot see existing values</p>
<p>First user obtains a lock at time of SELECT</p>	<pre>SELECT qty_on_hand FROM inventory WHERE partnum=4552 FOR UPDATE NOWAIT; ERROR: ORA-00054: resource busy and acquire with NOWAIT specified</pre>
<pre>UPDATE inventory SET qty_on_hand = 304 WHERE partnum=4552; COMMIT;</pre>	<p>Second user obtains lock, can see/change existing values</p>
<p>First user's lock released</p>	<pre>SELECT qty_on_hand FROM inventory WHERE partnum=4552 FOR UPDATE NOWAIT; QTY_ON_HAND ----- 304</pre>
	<pre>UPDATE inventory SET qty_on_hand = 303; COMMIT;</pre>

Flashback Queries (new 10g)

- View the value of data at a specific time in the past
- Must have sufficient privileges (Ch15)
- Oracle cannot flashback to earlier than the undo data available

```
SELECT * FROM writer;
INSERT INTO writer
VALUES ('6933', 'Gates', 'Bill', Null, Null, Null, Null, Null);
UPDATE writer SET ln = 'Colbert'
WHERE writerid = 'J525';
SELECT * FROM writer;
SELECT * FROM writer AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL '15' minute);
--show rows added or modified in past 15 minutes
SELECT * FROM writer
MINUS
SELECT * FROM writer AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL '15' minute);
--flip flop to show rows modified or deleted in past 15 minutes
SELECT * FROM writer AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL '15' minute)
MINUS
SELECT * FROM writer;
ROLLBACK;
```

Flashback Table (new 10g Enterprise) (Ch12)

- Restores a table's data to an earlier state
 - not available in Oracle Express Edition
 - ORA-00439: feature not enabled: Flashback Table
- Oracle cannot restore a table to a state earlier than the undo data available
- Oracle cannot restore a table to a state earlier than the most recent change to the table's **structure**
- Must have sufficient privileges (Ch15)

```
FLASHBACK TABLE writer
TO TIMESTAMP(SYSTIMESTAMP-INTERVAL '5' minute);
```

26