The COLUMN Command

Controls display of a column:

```
COL[UMN] [{column|alias} [option]]
```

- **CLE[AR]: Clears any column formats**
- **HEA[DING] *text*: Sets the column heading**
- **FOR[MAT] *format*: Changes the display of the column using a format model**
- **NOPRINT | PRINT**
- **NULL**

ORACLE

COLUMN Command Options

Option	Description
CLE[AR]	Clears any column formats
HEA[DING] *text*	Sets the column heading (a vertical line (|) forces a line feed in the heading if you do not use justification.)
FOR[MAT] *format*	Changes the display of the column data
NOPRI[NT]	Hides the column
NUL[L] *text*	Specifies text to be displayed for null values
PRI[NT]	Shows the column

Using the COLUMN Command

- **Create column headings.**

```
COLUMN last_name HEADING 'Employee|Name'
COLUMN salary JUSTIFY LEFT FORMAT $99,990.00
COLUMN manager FORMAT 999999999 NULL 'No manager'
```

- **Display the current setting for the LAST_NAME column.**

```
COLUMN last_name
```

- **Clear settings for the LAST_NAME column.**

```
COLUMN last_name CLEAR
```

ORACLE

7-19

Displaying or Clearing Settings

To show or clear the current COLUMN command settings, use the following commands:

Command	Description
COL[UMN] *column*	Displays the current settings for the specified column
COL[UMN]	Displays the current settings for all columns
COL[UMN] column CLE[AR]	Clears the settings for the specified column
CLE[AR] COL[UMN]	Clears the settings for all columns

COLUMN Format Models

Element	Description	Example	Result
9	Single zero-suppression digit	999999	1234
0	Enforces leading zero	099999	001234
$	Floating dollar sign	$9999	$1234
L	Local currency	L9999	L1234
.	Position of decimal point	9999.99	1234.00
,	Thousand separator	9,999	1,234

ORACLE

7-20

COLUMN Format Models

The slide displays sample COLUMN format models.

The Oracle Server displays a string of pound signs (#) in place of a whole number whose digits exceed the number of digits provided in the format model. It also displays pound signs in place of a value whose format model is alphanumeric but whose actual value is numeric.

Using the BREAK Command

Use the BREAK command to suppress duplicates

```
BREAK ON job_id
```

ORACLE

The BREAK Command

Use the BREAK command to divide rows into sections and suppress duplicate values. To ensure that the BREAK command works effectively, use the ORDER BY clause to order the columns that you are breaking on.

Syntax

```
BREAK on column[|alias|row]
```

In the syntax:

column[|alias|row	suppresses the display of duplicate values for a given column.

Clear all BREAK settings by using the CLEAR command:

```
CLEAR BREAK
```

Using the TTITLE and BTITLE Commands

- **Display headers and footers.**

```
TTI[TLE] [text|OFF|ON]
```

- **Set the report header.**

```
TTITLE 'Salary|Report'
```

- **Set the report footer.**

```
BTITLE 'Confidential'
```

ORACLE

7-22

The TTITLE and BTITLE Commands

Use the TTITLE command to format page headers and the BTITLE command for footers. Footers appear at the bottom of the page.

The syntax for BTITLE and TTITLE is identical. Only the syntax for TTITLE is shown. You can use the vertical bar (|) to split the text of the title across several lines.

Syntax

```
TTI[TLE]|BTI[TLE] [text|OFF|ON]
```

In the syntax:

text	represents the title text (enter single quotes if the text is more than one word).
OFF|ON	toggles the title either off or on. It is not visible when turned off.

The TTITLE example on the slide sets the report header to display Salary centered on one line and Report centered below it. The BTITLE example sets the report footer to display "Confidential." TTITLE automatically puts the date and a page number on the report.

Note: The slide gives an abridged syntax for TTITLE and BTITLE. Various options for TTITLE and BTITLE are covered in another SQL course.

8

Manipulating Data

Adding a New Row to a Table

DEPARTMENTS

70	Public Relations	100	1700

New row

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

Insert a new row into the `DEPARMENTS` table.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700
70	Public Relations	100	1700

ORACLE

The INSERT Statement Syntax

- Add new rows to a table by using the INSERT statement.

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

- Only one row is inserted at a time with this syntax.

Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the `INSERT` clause.

```
INSERT INTO departments(department_id, department_name,
                        manager_id, location_id)
VALUES      (70, 'Public Relations', 100, 1700);
1 row created.
```

- Enclose character and date values within single quotation marks.

Inserting Rows with Null Values

- **Implicit method: Omit the column from the column list.**

```
INSERT INTO   departments (department_id,
                           department_name    )
VALUES        (30, 'Purchasing');
1 row created.
```

- **Explicit method: Specify the `NULL` keyword in the `VALUES` clause.**

```
INSERT INTO   departments
VALUES        (100, 'Finance', NULL, NULL);
1 row created.
```

Inserting Special Values

The `SYSDATE` function records the current date and time.

```
INSERT INTO employees (employee_id,
                 first_name, last_name,
                 email, phone_number,
                 hire_date, job_id, salary,
                 commission_pct, manager_id,
                 department_id)
VALUES           (113,
                 'Louis', 'Popp',
                 'LPOPP', '515.124.4567',
                 SYSDATE, 'AC_ACCOUNT', 6900,
                 NULL, 205, 100);
1 row created.
```

Inserting Specific Date Values

- **Add a new employee.**

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
             'AC_ACCOUNT', 11000, NULL, 100, 30);
1 row created.
```

- **Verify your addition.**

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	

ORACLE

Creating a Script

- Use & substitution in a SQL statement to prompt for values.
- & is a placeholder for the variable value.

```
INSERT INTO departments
            (department_id, department_name, location_id)
VALUES      (&department_id, '&department_name',&location);
```

Define Substitution Variables

"department_id"	40
"department_name"	Human Resources
"location"	2500

```
1 row created.
```

Copying Rows from Another Table

- Write your `INSERT` statement with a subquery.

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';
4 rows created.
```

- Do not use the `VALUES` clause.
- Match the number of columns in the `INSERT` clause to those in the subquery.

Changing Data in a Table

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMM
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	60	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	60	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	60	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

Update rows in the `EMPLOYEES` table.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMM
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	**30**	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	**30**	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	**30**	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

ORACLE

The UPDATE Statement Syntax

- Modify existing rows with the UPDATE statement.

```
UPDATE        table
SET           column = value [, column = value, ...]
[WHERE        condition];
```

- Update more than one row at a time, if required.

Updating Rows in a Table

- Specific row or rows are modified if you specify the `WHERE` clause.

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- All rows in the table are modified if you omit the `WHERE` clause.

```
UPDATE   copy_emp
SET      department_id = 110;
22 rows updated.
```

ORACLE

Updating Two Columns with a Subquery

Update employee 114's job and department to match that of employee 205.

```
UPDATE   employees
SET      job_id  = (SELECT  job_id
                    FROM    employees
                    WHERE   employee_id = 205),
         salary  = (SELECT  salary
                    FROM    employees
                    WHERE   employee_id = 205)
WHERE    employee_id    =  114;
1 row updated.
```

Updating Rows Based on Another Table

Use subqueries in `UPDATE` statements to update rows in a table based on values from another table.

```
UPDATE   copy_emp
SET      department_id  =  (SELECT department_id
                            FROM employees
                            WHERE employee_id = 100)
WHERE    job_id         =  (SELECT job_id
                            FROM employees
                            WHERE employee_id = 200);
1 row updated.
```

Updating Rows: Integrity Constraint Error

```
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110;
```

```
UPDATE employees
       *
ERROR at line 1:
ORA-02291: integrity constraint
(HR.EMP_DEPT_FK)  violated - parent key not
found
```

Removing a Row from a Table

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
70	Public Relations	100	1700
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400
100	Finance		
80	Sales	149	2500

Delete a row from the DEPARTMENTS table.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
70	Public Relations	100	1700
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

ORACLE

8-18

Removing a Row from a Table

The graphic in the slide removes the Finance department from the DEPARTMENTS table (assuming that there are no constraints defined on the DEPARTMENTS table).

The DELETE Statement

You can remove existing rows from a table by using the DELETE statement.

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

Deleting Rows from a Table

- Specific rows are deleted if you specify the `WHERE` clause.

```
 DELETE FROM departments
 WHERE  department_name = 'Finance';
1 row deleted.
```

- All rows in the table are deleted if you omit the `WHERE` clause.

```
DELETE FROM  copy_emp;
22 rows deleted.
```

Deleting Rows Based on Another Table

Use subqueries in `DELETE` statements to remove rows from a table based on values from another table.

```
DELETE FROM employees
WHERE  department_id =
                       (SELECT department_id
                        FROM   departments
                        WHERE  department_name LIKE '%Public%');
1 row deleted.
```

Deleting Rows: Integrity Constraint Error

```
DELETE FROM departments
WHERE       department_id = 60;
```

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

```
DELETE FROM departments
            *
ERROR at line 1:
ORA-02292: integrity constraint
(HR.EMP_DEPT_FK)  violated - child record found
```

ORACLE

Using a Subquery in an INSERT Statement

```
INSERT INTO
        (SELECT employee_id, last_name,
                email, hire_date, job_id, salary,
                department_id
         FROM   employees
         WHERE  department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ST_CLERK', 5000, 50);

1 row created.
```

Using a Subquery in an `INSERT` Statement

```
SELECT employee_id, last_name, email, hire_date,
       job_id, salary, department_id
FROM   employees
WHERE  department_id = 50;
```

EMPLOYEE_ID	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
124	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50
141	Rajs	TRAJS	17-OCT-95	ST_CLERK	3500	50
142	Davies	CDAVIES	29-JAN-97	ST_CLERK	3100	50
143	Matos	RMATOS	15-MAR-98	ST_CLERK	2600	50
144	Vargas	PVARGAS	09-JUL-98	ST_CLERK	2500	50
99999	Taylor	DTAYLOR	07-JUN-99	ST_CLERK	5000	50

6 rows selected.

Overview of the Explict Default Feature

- With the explicit default feature, you can use the `DEFAULT` keyword as a column value where the column default is desired.
- The addition of this feature is for compliance with the SQL: 1999 Standard.
- This allows the user to control where and when the default value should be applied to data.
- Explicit defaults can be used in `INSERT` and `UPDATE` statements.

ORACLE

8-26

Explicit Defaults

The `DEFAULT` keyword can be used in `INSERT` and `UPDATE` statements to identify a default column value. If no default value exists, a null value is used.

Using Explicit Default Values

- **DEFAULT with INSERT:**

```
INSERT INTO departments
  (department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```

- **DEFAULT with UPDATE:**

```
UPDATE departments
SET manager_id = DEFAULT WHERE department_id = 10;
```

ORACLE

The `MERGE` Statement

- Provides the ability to conditionally update or insert data into a database table
- Performs an `UPDATE` if the row exists and an `INSERT` if it is a new row:
  - Avoids separate updates
  - Increases performance and ease of use
  - Is useful in data warehousing applications

ORACLE

MERGE Statement Syntax

You can conditionally insert or update rows in a table by using the MERGE statement.

```
MERGE INTO table_name AS table_alias
  USING (table|view|sub_query) AS alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
    col1 = col_val1,
    col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

Merging Rows

Insert or update rows in the `COPY_EMP` table to match the `EMPLOYEES` table.

```
MERGE INTO copy_emp AS c
  USING employees e
  ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
  UPDATE SET
     c.first_name     = e.first_name,
     c.last_name      = e.last_name,
     ...
     c.department_id  = e.department_id
WHEN NOT MATCHED THEN
 INSERT VALUES(e.employee_id, e.first_name, e.last_name,
          e.email, e.phone_number, e.hire_date, e.job_id,
          e.salary, e.commission_pct, e.manager_id,
          e.department_id);
```

ORACLE

Merging Rows

```
SELECT *
FROM COPY_EMP;

no rows selected
```

```
MERGE INTO copy_emp c
  USING employees e
  ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
  UPDATE SET
     ...
WHEN NOT MATCHED THEN
 INSERT VALUES...;
```

```
SELECT *
FROM COPY_EMP;

20 rows selected.
```

Database Transactions

A database transaction consists of one of the following:

- **DML statements which constitute one consistent change to the data**
- **One DDL statement**
- **One DCL statement**

ORACLE

Database Transactions

- Begin when the first DML SQL statement is executed
- End with one of the following events:
  - A `COMMIT` or `ROLLBACK` statement is issued
  - A DDL or DCL statement executes (automatic commit)
  - The user exits *i*SQL*Plus
  - The system crashes

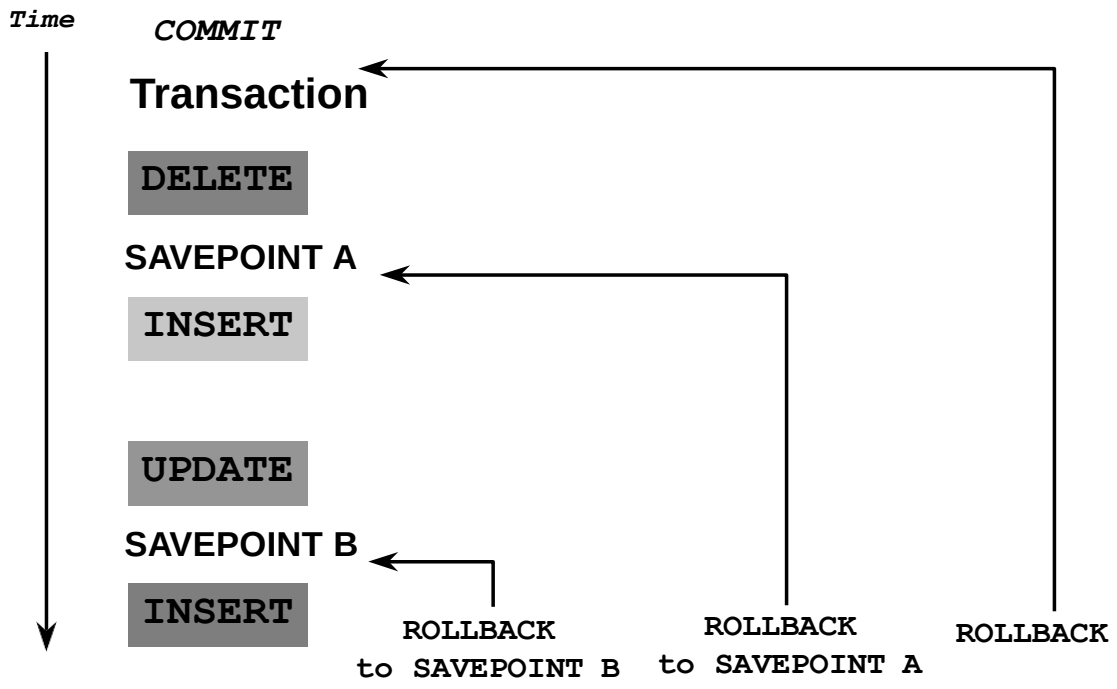
Advantages of `COMMIT` and `ROLLBACK` Statements

With `COMMIT` and `ROLLBACK` statements, you can:

- **Ensure data consistency**
- **Preview data changes before making changes permanent**
- **Group logically related operations**

ORACLE

Controlling Transactions



Rolling Back Changes to a Marker

- Create a marker in a current transaction by using the `SAVEPOINT` statement.
- Roll back to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

```
UPDATE...
SAVEPOINT update_done;
Savepoint created.
INSERT...
ROLLBACK TO update_done;
Rollback complete.
```

Implicit Transaction Processing

- An automatic commit occurs under the following circumstances:
  - DDL statement is issued
  - DCL statement is issued
  - Normal exit from *i*SQL*Plus, without explicitly issuing `COMMIT` or `ROLLBACK` statements
- An automatic rollback occurs under an abnormal termination of *i*SQL*Plus or a system failure.

ORACLE

State of the Data Before `COMMIT` or `ROLLBACK`

- The previous state of the data can be recovered.
- The current user can review the results of the DML operations by using the `SELECT` statement.
- Other users *cannot* view the results of the DML statements by the current user.
- The affected rows are *locked*; other users cannot change the data within the affected rows.

ORACLE

State of the Data After `COMMIT`

- Data changes are made permanent in the database.
- The previous state of the data is permanently lost.
- All users can view the results.
- Locks on the affected rows are released; those rows are available for other users to manipulate.
- All savepoints are erased.

ORACLE

Committing Data

- **Make the changes.**

```
DELETE FROM employees
WHERE  employee_id = 99999;
1 row deleted.

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 row inserted.
```

- **Commit the changes.**

```
COMMIT;
Commit complete.
```

State of the Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

```
DELETE FROM  copy_emp;
22 rows deleted.
ROLLBACK;
Rollback complete.
```

ORACLE

Statement-Level Rollback

- If a single DML statement fails during execution, only that statement is rolled back.
- The Oracle Server implements an implicit savepoint.
- All other changes are retained.
- The user should terminate transactions explicitly by executing a `COMMIT` or `ROLLBACK` statement.

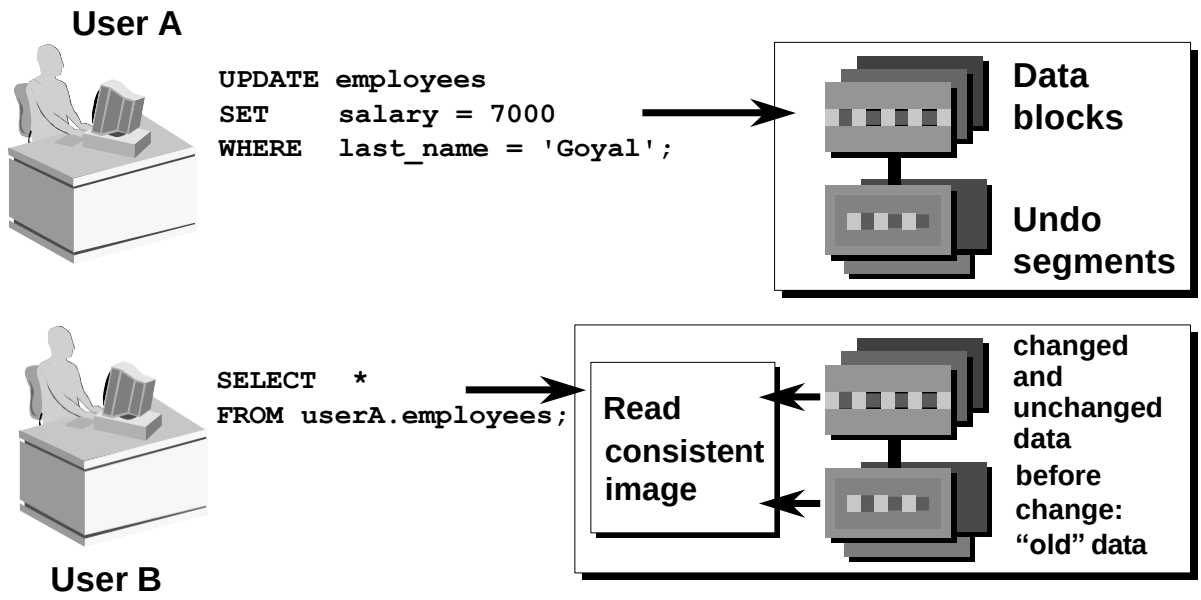
ORACLE

Read Consistency

- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with changes made by another user.
- Read consistency ensures that on the same data:
  - Readers do not wait for writers
  - Writers do not wait for readers

ORACLE

Implementation of Read Consistency



Locking

In an Oracle database, locks:

- Prevent destructive interaction between concurrent transactions
- Require no user action
- Use the lowest level of restrictiveness
- Are held for the duration of the transaction
- Are of two types: explicit locking and implicit locking

ORACLE

Implicit Locking

- Two lock modes:
  - Exclusive: Locks out other users
  - Share: Allows other users to access the server
- High level of data concurrency:
  - DML: Table share, row exclusive
  - Queries: No locks required
  - DDL: Protects object definitions
- Locks held until commit or rollback

ORACLE

Read Consistency Example

Output	Time	Session 1	Session 2
24000	t1	`SELECT salary FROM employees WHERE last_name='King';`	
	t2		`UPDATE employees SET salary=salary+10000 WHERE last_name='King';`
24000	t3	`SELECT salary FROM employees WHERE last_name='King';`	
	t4		`COMMIT;`
34000	t5	`SELECT salary FROM employees WHERE last_name='King';`	

9

Creating and Managing Tables

ORACLE

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows and columns
View	Logically represents subsets of data from one or more tables
Sequence	Numeric value generator
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Naming Rules

Table names and column names:

- **Must begin with a letter**
- **Must be 1 to 30 characters long**
- **Must contain only A–Z, a–z, 0–9, _, $, and #**
- **Must not duplicate the name of another object owned by the same user**
- **Must not be an Oracle Server reserved word**

ORACLE

The CREATE TABLE Statement

- You must have:
  - CREATE TABLE privilege
  - A storage area

```
CREATE TABLE [schema.]table
             (column datatype [DEFAULT expr][, ...]);
```

- You specify:
  - Table name
  - Column name, column data type, and column size

Referencing Another User's Tables

- Tables belonging to other users are not in the user's schema.
- You should use the owner's name as a prefix to those tables.

The `DEFAULT` Option

- Specify a default value for a column during an `INSERT` operation.

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- Literal values, expressions, or SQL functions are legal values.
- Another column's name or a pseudocolumn are illegal values.
- The default data type must match the column data type.

ORACLE

Creating Tables

- **Create the table.**

```
CREATE TABLE dept
            (deptno NUMBER(2),
             dname   VARCHAR2(14),
             loc     VARCHAR2(13));
Table created.
```

- **Confirm creation of the table.**

```
DESCRIBE dept
```

Name	Null?	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

ORACLE

Tables in the Oracle Database

- **User tables:**
  - **Are a collection of tables created and maintained by the user**
  - **Contain user information**
- **Data dictionary:**
  - **Is a collection of tables created and maintained by the Oracle Server**
  - **Contain database information**

ORACLE

Querying the Data Dictionary

- See the names of tables owned by the user.

```
SELECT   table_name
FROM user_tables;
```

- View distinct object types owned by the user.

```
SELECT DISTINCT object_type
FROM user_objects;
```

- View tables, views, synonyms, and sequences owned by the user.

```
SELECT *
FROM user_catalog;
```

ORACLE

Creating a Table by Using a Subquery Syntax

- Create a table and insert rows by combining the `CREATE TABLE` statement and the `AS` *`subquery`* option.

```
CREATE TABLE table
       [(column, column...)]
AS subquery;
```

- Match the number of specified columns to the number of subquery columns.
- Define columns with column names and default values.

Creating a Table by Using a Subquery

```
CREATE TABLE dept80
  AS
    SELECT  employee_id, last_name,
            salary*12 ANNSAL,
            hire_date
    FROM    employees
    WHERE   department_id = 80;
Table created.
```

```
 DESCRIBE dept80
```

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

The `ALTER TABLE` Statement

Use the `ALTER TABLE` statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

ORACLE

The ALTER TABLE Statement

Use the ALTER TABLE statement to add, modify or drop columns.

```
ALTER TABLE table
ADD         (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
MODIFY      (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
DROP       (column);
```

ORACLE

Adding a Column

DEPT80

EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
149	Zlotkey	126000	29-JAN-00
174	Abel	132000	11-MAY-96
176	Taylor	103200	24-MAR-98

New column

JOB_ID

Add a new column to the `DEPT80` table.

DEPT80

EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
149	Zlotkey	126000	29-JAN-00	
174	Abel	132000	11-MAY-96	
176	Taylor	103200	24-MAR-98	

ORACLE

Adding a Column

- Use the `ADD` clause to add columns.

```
ALTER TABLE dept80
ADD         (job_id VARCHAR2(9));
Table altered.
```

- The new column becomes the last column.

EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
149	Zlotkey	126000	29-JAN-00	
174	Abel	132000	11-MAY-96	
176	Taylor	103200	24-MAR-98	

ORACLE

Modifying a Column

- You can change a column's data type, size, and default value.

```
ALTER TABLE   dept80
MODIFY        (last_name VARCHAR2(30));
Table altered.
```

- A change to the default value affects only subsequent insertions to the table.

Dropping a Column

Use the `DROP COLUMN` clause to drop columns you no longer need from the table.

```
ALTER TABLE  dept80
DROP  COLUMN   job_id;
Table altered.
```

The SET UNUSED Option

- You use the SET UNUSED option to mark one or more columns as unused.
- You use the DROP UNUSED COLUMNS option to remove the columns that are marked as unused.

```
ALTER TABLE    table
SET   UNUSED  (column);
          OR
ALTER      TABLE  table
SET UNUSED COLUMN column;
```

```
ALTER TABLE table
DROP  UNUSED COLUMNS;
```

ORACLE

Dropping a Table

- All data and structure in the table is deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- You *cannot* roll back the `DROP TABLE` statement.

```
DROP TABLE dept80;
Table dropped.
```

Changing the Name of an Object

- To change the name of a table, view, sequence, or synonym, execute the `RENAME` statement.

```
RENAME dept TO detail_dept;
Table renamed.
```

- You must be the owner of the object.

Truncating a Table

- The `TRUNCATE TABLE` statement:
  - Removes all rows from a table
  - Releases the storage space used by that table

```
TRUNCATE TABLE detail_dept;
Table truncated.
```

- You cannot roll back row removal when using `TRUNCATE`.
- Alternatively, you can remove rows by using the `DELETE` statement.

Adding Comments to a Table

- You can add comments to a table or column by using the `COMMENT` statement.

```
COMMENT ON TABLE employees
IS 'Employee Information';
Comment created.
```

- Comments can be viewed through the data dictionary views:
  - `ALL_COL_COMMENTS`
  - `USER_COL_COMMENTS`
  - `ALL_TAB_COMMENTS`
  - `USER_TAB_COMMENTS`

10

Including Constraints

What Are Constraints?

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies.
- The following constraint types are valid:
  - `NOT NULL`
  - `UNIQUE`
  - `PRIMARY KEY`
  - `FOREIGN KEY`
  - `CHECK`

ORACLE

Constraint Guidelines

- Name a constraint or the Oracle server generates a name by using the `SYS_C`*`n`* format.
- Create a constraint either:
  - At the same time as the table is created, or
  - After the table has been created.
- Define a constraint at the column or table level.
- View a constraint in the data dictionary.

Defining Constraints

```
CREATE TABLE [schema.]table
             (column datatype [DEFAULT expr]
             [column_constraint],
             ...
             [table_constraint][,...]);
```

```
CREATE TABLE employees(
             employee_id  NUMBER(6),
             first_name   VARCHAR2(20),
             ...
             job_id       VARCHAR2(10) NOT NULL,
             CONSTRAINT emp_emp_id_pk
                          PRIMARY KEY (EMPLOYEE_ID));
```

Defining Constraints

- **Column constraint level:**

```
column [CONSTRAINT constraint_name] constraint_type,
```

- **Table constraint level:**

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

The NOT NULL Constraint

Ensures that null values are not permitted for the column

100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000		
101	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000		100
102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000		100
103	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000		102
104	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000		103
205	[illegible]	[illegible]	[illegible]	[illegible]-94	[illegible]	[illegible]		[illegible]
206	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300		205

20 rows selected.

NOT NULL constraint (No row can contain a null value for this column.)

NOT NULL constraint

Absence of NOT NULL constraint (Any row can contain null for this column.)

ORACLE

The NOT NULL Constraint

Is defined at the column level

```
CREATE TABLE employees(
    employee_id    NUMBER(6),
    last_name      VARCHAR2(25) NOT NULL,
    salary         NUMBER(8,2),
    commission_pct NUMBER(2,2),
    hire_date      DATE
                   CONSTRAINT emp_hire_date_nn
                   NOT NULL,
...
```

System named

User named

The UNIQUE Constraint

UNIQUE constraint

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	EMAIL
100	King	SKING
101	Kochhar	NKOCHHAR
102	De Haan	LDEHAAN
103	Hunold	AHUNOLD
104	Ernst	BERNST
107	Lorentz	DLORENTZ

INSERT INTO

208	Smith	JSMITH	**Allowed**
209	Smith	JSMITH	**Not allowed: already exists**

The UNIQUE Constraint

Is defined at either the table level or the column level

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    CONSTRAINT emp_email_uk UNIQUE(email));
```

The PRIMARY KEY Constraint

DEPARTMENTS

PRIMARY KEY

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

Not allowed (null value)

INSERT INTO

	Public Accounting		1400
50	Finance	124	1500

Not allowed (50 already exists)

The PRIMARY KEY Constraint

Is defined at either the table level or the column level

```
CREATE TABLE   departments(
    department_id        NUMBER(4),
    department_name      VARCHAR2(30)
      CONSTRAINT dept_name_nn NOT NULL,
    manager_id           NUMBER(6),
    location_id          NUMBER(4),
      CONSTRAINT dept_id_pk PRIMARY KEY(department_id));
```

The FOREIGN KEY Constraint

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

PRIMARY KEY

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60

FOREIGN KEY

INSERT INTO

210	Ford	9
211	Ford	60

Not allowed (9 does not exist)

Allowed

ORACLE

The FOREIGN KEY Constraint

Is defined at either the table level or the column level

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    department_id    NUMBER(4),
    CONSTRAINT emp_dept_fk FOREIGN KEY
      (department_id)  REFERENCES
      departments(department_id),
    CONSTRAINT emp_email_uk UNIQUE(email));
```

`FOREIGN KEY` Constraint Keywords

- `FOREIGN KEY`: Defines the column in the child table at the table constraint level
- `REFERENCES`: Identifies the table and column in the parent table
- `ON DELETE CASCADE`: Deletes the dependent rows in the child table when a row in the parent table is deleted
- `ON DELETE SET NULL`: Converts dependent foreign key values to null

ORACLE

The CHECK Constraint

- Defines a condition that each row must satisfy
- The following expressions are not allowed:
  - References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
  - Calls to SYSDATE, UID, USER, and USERENV functions
  - Queries that refer to other values in other rows

```
..., salary  NUMBER(2)
     CONSTRAINT emp_salary_min
            CHECK (salary > 0),...
```

ORACLE

Adding a Constraint Syntax

Use the `ALTER TABLE` statement to:

- Add or drop a constraint, but not modify its structure
- Enable or disable constraints
- Add a `NOT NULL` constraint by using the `MODIFY` clause

```
ALTER TABLE table
ADD [CONSTRAINT constraint] type (column);
```

Adding a Constraint

Add a `FOREIGN KEY` constraint to the `EMPLOYEES` table to indicate that a manager must already exist as a valid employee in the `EMPLOYEES` table.

```
ALTER TABLE     employees
ADD CONSTRAINT  emp_manager_fk
  FOREIGN KEY(manager_id)
  REFERENCES employees(employee_id);
Table altered.
```

Dropping a Constraint

- Remove the manager constraint from the `EMPLOYEES` table.

```
ALTER TABLE      employees
DROP CONSTRAINT  emp_manager_fk;
Table altered.
```

- Remove the `PRIMARY KEY` constraint on the `DEPARTMENTS` table and drop the associated `FOREIGN KEY` constraint on the `EMPLOYEES.DEPARTMENT_ID` column.

```
ALTER TABLE  departments
DROP PRIMARY KEY CASCADE;
Table altered.
```

ORACLE

Cascading Constraints

- The `CASCADE CONSTRAINTS` clause is used along with the `DROP COLUMN` clause.
- The `CASCADE CONSTRAINTS` clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.
- The `CASCADE CONSTRAINTS` clause also drops all multicolumn constraints defined on the dropped columns.

ORACLE

Cascading Constraints

Example

```
ALTER TABLE test1
DROP (pk) CASCADE CONSTRAINTS;
Table altered.
```

```
ALTER TABLE test1
DROP (pk, fk, col1) CASCADE CONSTRAINTS;
Table altered.
```

ORACLE

Viewing Constraints

Query the `USER_CONSTRAINTS` table to view all constraint definitions and names.

```
SELECT   constraint_name, constraint_type,
         search_condition
FROM     user_constraints
WHERE    table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	C	SEARCH_CONDITION
EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL
EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL
EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL
EMP_JOB_NN	C	"JOB_ID" IS NOT NULL
EMP_SALARY_MIN	C	salary > 0
EMP_EMAIL_UK	U	
EMP_EMP_ID_PK	P	
EMP_DEPT_FK	R	

ORACLE

Viewing the Columns Associated with Constraints

View the columns associated with the constraint names in the `USER_CONS_COLUMNS` view.

```
SELECT   constraint_name, column_name
FROM     user_cons_columns
WHERE    table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	COLUMN_NAME
EMP_DEPT_FK	DEPARTMENT_ID
EMP_EMAIL_NN	EMAIL
EMP_EMAIL_UK	EMAIL
EMP_EMP_ID_PK	EMPLOYEE_ID
EMP_HIRE_DATE_NN	HIRE_DATE
EMP_JOB_FK	JOB_ID
EMP_JOB_NN	JOB_ID
EMP_LAST_NAME_NN	LAST_NAME
EMP_MANAGER_FK	MANAGER_ID
EMP_SALARY_MIN	SALARY

ORACLE