



Structured Query Language (SQL)

Asif Sohail

University of the Punjab
Punjab University College of Information Technology (PUCIT)

Note:

The slides are adapted from “Oracle Corporation”

Introduction

- SQL pronounced as **Sequel** is a language for RDBMS or SQL makes RDBMS possible.
- It was first defined by Chamberlin and others at IBM's research laboratory in San Jose, California (Late 1970's)
- ANSI and ISO have worked for the standardization of SQL.
- SQL is a **non procedural language**.
- SQL does much more and is more powerful than just asking queries from the database.
- Using SQL, one can:

Introduction

- Create Relations or Tables.
- Insert, Delete and Update Records.
- Apply Validation Checks.
- Create users.
- Apply Security Restrictions etc. etc.

Communicating with a RDBMS Using SQL

**SQL statement
is entered**

```
SQL> SELECT loc  
2   FROM dept;
```

**Statement is sent to
database**

Database

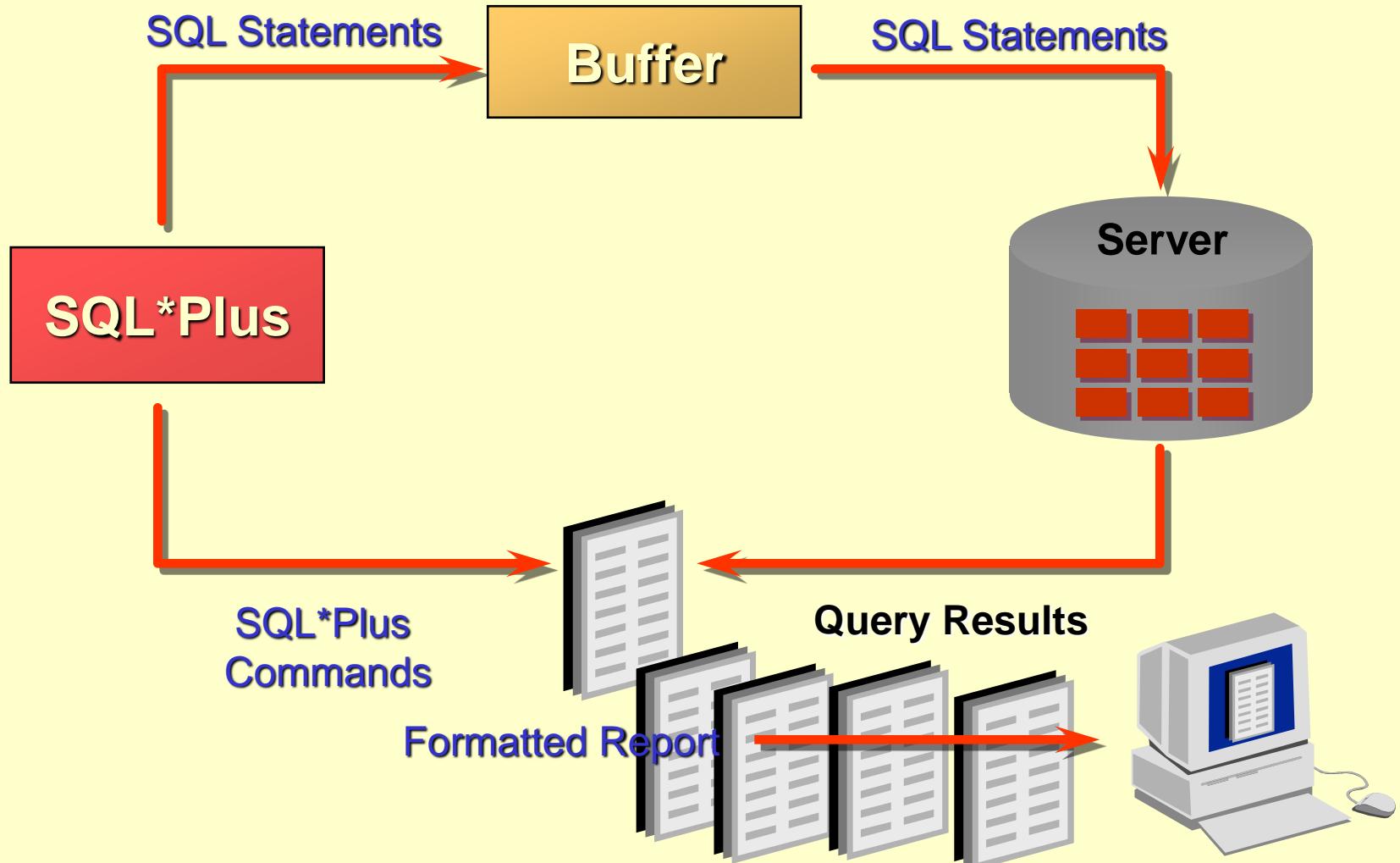
Data is displayed

LOC

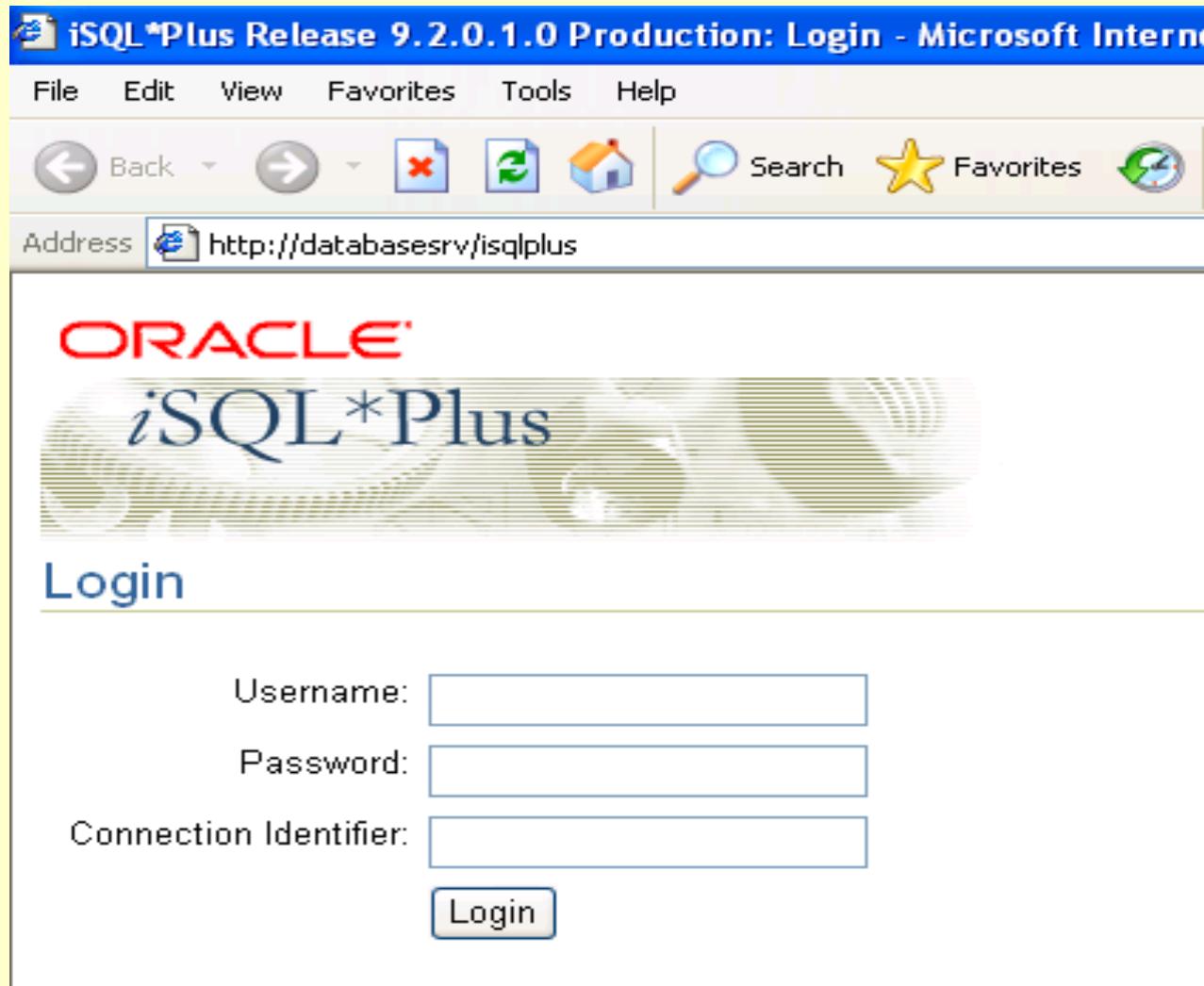
NEW YORK
DALLAS
CHICAGO
BOSTON



SQL and SQL Plus Interaction



Logging In to SQL Plus



SQL Statements

SELECT

Data retrieval

INSERT

UPDATE

DELETE

Data manipulation language (DML)

CREATE

ALTER

DROP

RENAME

TRUNCATE

Data definition language (DDL)

COMMIT

ROLLBACK

SAVEPOINT

Transaction control

GRANT

REVOKE

Data control language (DCL)

Tables Used in the Course

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		17-NOV-81	5000		10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7900	JAMES	CLERK	7698	03-DEC-81	950		30

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

Querying the Tables

- The Select statement is the building block for querying the tables:
- Syntax:

```
SELECT [DISTINCT] {*} | column [alias],...}  
FROM    table (s)  
[WHERE clause]  
[ORDER BY clause]  
[GROUP BY clause]  
[HAVING clause]
```
- **SELECT** identifies *what* columns
- **FROM** identifies *which* table

Capabilities of SQL SELECT Statements

Selection

Table 1

Projection

Table 1

Join

Table 1

Table 2

Writing SQL Statements

- SQL statements are not case sensitive.
- SQL statements can be on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Tabs and indents are used to enhance readability.

Writing SQL Statements

- Examples (Querying emp table):
 - Select * from emp;
 - Select empno from emp;
 - Select empno,ename,job from emp;
 - Select ename,sal+100 from emp;
 - Select ename, sal*2/100 “Tax” from emp;
 - Select empno,ename,job,sal,sal*12 “Annual Sal” from emp;
 - Select distinct job from emp;

Selecting All Columns

```
SQL> SELECT *  
2  FROM dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Selecting Specific Columns

```
SQL> SELECT deptno, loc  
2   FROM dept;
```

DEPTNO	LOC
10	NEW YORK
20	DALLAS
30	CHICAGO
40	BOSTON

Editing SQL Statements

- To edit the last SQL statement and then execute it, typed at SQL prompt and press Enter Key.
- Notepad will be opened with the last SQL statement written in it.
- Edit and statement, save it and exit from the notepad.
- At SQL prompt, type / and press enter to execute the statement.

Arithmetic Expressions

- Create expressions on NUMBER and DATE data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic Operators

```
SQL> SELECT ename, sal, sal+300  
2   FROM emp;
```

ENAME	SAL	SAL+300
KING	5000	5300
BLAKE	2850	3150
CLARK	2450	2750
JONES	2975	3275
MARTIN	1250	1550
ALLEN	1600	1900
...		
14 rows selected.		

Operator Precedence



- Multiplication and division take priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to clarify statements.

Defining a Column Alias

- Renames a column heading
- Is useful with calculations
- Immediately follows column name; optional AS keyword between column name and alias
- Requires double quotation marks if it contains spaces or special characters or is case sensitive

Using Column Aliases

```
SQL> SELECT ename AS name, sal salary  
2  FROM emp;
```

NAME	SALARY
-----	-----
...	

```
SQL> SELECT ename "Name",  
2          sal*12 "Annual Salary"  
3  FROM emp;
```

Name	Annual Salary
-----	-----
...	

Defining a Null Value

- A null is a value that is unavailable, unassigned, unknown, or inapplicable.
- A null is not the same as zero or a blank space.

```
SQL> SELECT      ename,  job,  comm  
2   FROM        emp;
```

ENAME	JOB	COMM
KING	PRESIDENT	
BLAKE	MANAGER	
...		
TURNER	SALESMAN	0
...		
14 rows selected.		

Null Values in Arithmetic Expressions

- Arithmetic expressions containing a null value evaluate to null.

```
SQL> select ename, 12*sal+comm  
2  from emp;
```

ENAME	12*SAL+COMM
KING	

To avoid the above, use nvl function.

ENAME	12*SAL+ NVL(COMM, 0)
KING	5000

Concatenation Operator

- Concatenates columns or character strings to other columns
- Is represented by two vertical bars (||)
- Creates a resultant column that is a character expression

```
SQL> SELECT    ename || job AS "Employees"  
2   FROM      emp;
```

Literal Character Strings

- A literal is a character, expression, or number included in the SELECT list.
- Date and character literal values must be enclosed within single quotation marks.
- Each character string is output once for each row returned.

```
SQL> SELECT ename || ' is a ' || job
2          AS "Employee Details"
3  FROM    emp;
```

Duplicate Rows

- The default display of queries is all rows, including duplicate rows.

```
SQL> SELECT deptno  
2   FROM emp;
```

DEPTNO

10
30
10
20
...
14 rows selected.

Eliminating Duplicate Rows

Eliminate duplicate rows by using the **DISTINCT** keyword in the **SELECT** clause.

```
SQL> SELECT DISTINCT deptno  
2   FROM emp;
```

DEPTNO

10
20
30



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



(SQL)

Restricting and Sorting Data

Asif Sohail

**University of the Punjab
Punjab University College of Information Technology (PUCIT)**

Objectives

After completing this lesson, you should be able to do the following:

- Limit the rows retrieved by a query
- Sort the rows retrieved by a query

Limiting Rows Using a Selection

EMP

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		10
7566	JONES	MANAGER		20
...				

**"...retrieve all
employees
in department 10"**

EMP

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7782	CLARK	MANAGER		10
7934	MILLER	CLERK		10

Limiting Rows Selected

- Restrict the rows returned by using the WHERE clause.

```
SELECT      [DISTINCT]  [* | column [alias], ...]
FROM        table
[WHERE      condition(s)] ;
```

- The WHERE clause follows the FROM clause.

```
SQL> SELECT ename, job, deptno
  2  FROM emp
  3 WHERE job='CLERK' ;
```

Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Character Strings and Dates

- Character strings and date values are enclosed in single quotation marks.
- Character values are case sensitive and date values are format sensitive.
- The default date format is DD-MON-YY or mm/dd/yyyy .

```
SQL> SELECT    ename, job, deptno  
  2  FROM      emp  
  3  WHERE     ename = 'JAMES' ;
```

Logical Operators

Operator	Meaning
AND	Returns TRUE if <i>both</i> component conditions are TRUE
OR	Returns TRUE if <i>either</i> component condition is TRUE
NOT	Returns TRUE if the following condition is FALSE

Using the AND Operator

AND requires both conditions to be TRUE.

```
SQL> SELECT empno, ename, job, sal  
2   FROM emp  
3 WHERE sal>=1100  
4 AND job='CLERK' ;
```

EMPNO	ENAME	JOB	SAL
7876	ADAMS	CLERK	1100
7934	MILLER	CLERK	1300

Using the OR Operator

OR requires either condition to be TRUE.

```
SQL> SELECT empno, ename, job, sal  
  2  FROM    emp  
  3 WHERE   sal>=1100  
  4 OR      job='CLERK' ;
```

EMPNO	ENAME	JOB	SAL
7839	KING	PRESIDENT	5000
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7566	JONES	MANAGER	2975
7654	MARTIN	SALESMAN	1250
...			
7900	JAMES	CLERK	950
...			

14 rows selected.

Rules of Precedence

Order Evaluated	Operator
1	All comparison operators
2	NOT
3	AND
4	OR

- Override rules of precedence by using parentheses.

Rules of Precedence

```
SQL> SELECT ename, job, sal  
2   FROM emp  
3 WHERE job='SALESMAN'  
4 OR  job='PRESIDENT'  
5 AND sal>1500;
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
MARTIN	SALESMAN	1250
ALLEN	SALESMAN	1600
TURNER	SALESMAN	1500
WARD	SALESMAN	1250

Rules of Precedence

Use parentheses to force priority.

```
SQL> SELECT      ename, job, sal  
  2  FROM       emp  
  3 WHERE      (job=' SALESMAN '  
  4 OR        job=' PRESIDENT ' )  
  5 AND      sal>1500 ;
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
ALLEN	SALESMAN	1600

Other Comparison Operators / Special Functions

Operator	Meaning
BETWEEN ...AND...	Between two values (inclusive)
IN(list)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Using the BETWEEN Operator

- Use the BETWEEN operator to display rows based on a range of values.

```
SQL> SELECT ename, sal  
2  FROM emp  
3  WHERE sal BETWEEN 1000 AND 1500;
```

ENAME	SAL	Lower limit	Higher limit
MARTIN	1250		
TURNER	1500		
WARD	1250		
ADAMS	1100		
MILLER	1300		

Using the IN Operator

- Use the IN operator to test for values in a list.

```
SQL> SELECT empno, ename, sal, mgr  
  2  FROM emp  
  3  WHERE mgr IN (7902, 7566, 7788);
```

EMPNO	ENAME	SAL	MGR
7902	FORD	3000	7566
7369	SMITH	800	7902
7788	SCOTT	3000	7566
7876	ADAMS	1100	7788

Using the LIKE Operator

- Use the LIKE operator to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers.
 - % denotes zero or many characters.
 - _ denotes one character.

```
SQL> SELECT    ename
      2  FROM     emp
      3 WHERE    ename LIKE 'S%' ;
```

```
SQL> SELECT    ename
      2  FROM     emp
      3 WHERE    ename LIKE '_A%' ;
```

Using the IS NULL Operator

- Test for null values with the IS NULL operator.

```
SQL> SELECT    ename, mgr  
  2  FROM      emp  
  3  WHERE     mgr IS NULL;
```

ENAME	MGR
KING	

```
SQL> SELECT    empno, ename, comm  
  2  FROM      emp  
  3  WHERE     comm IS NOT NULL;
```

Rules of Precedence

Order Evaluated	Operator
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT logical condition
7	AND logical condition
8	OR logical condition

- Override rules of precedence by using parentheses.

ORDER BY Clause

- Sort rows with the ORDER BY clause
 - ASC: ascending order, default
 - DESC: descending order
- The ORDER BY clause comes last in the SELECT statement.

```
SELECT      [DISTINCT] { * | column [alias], ... }  
FROM        table  
[WHERE       condition(s) ]  
[ORDER BY    {column, expr, alias} [ASC|DESC]] ;
```

ORDER BY Clause

```
SQL> SELECT      ename, job, deptno, hiredate  
2   FROM        emp  
3   ORDER BY hiredate;
```

```
SQL> SELECT      ename, job, deptno, hiredate  
2   FROM        emp  
3   ORDER BY hiredate DESC;
```

```
SQL> SELECT      empno, ename, sal*12 annsal  
2   FROM        emp  
3   ORDER BY annsal;
```

```
SQL> SELECT      ename, deptno, sal  
2   FROM        emp  
3   ORDER BY deptno, sal DESC;
```

Sorting by Column Alias

```
SQL> SELECT      emno, ename, sal*12 annsal  
  2  FROM        emp  
  3  ORDER BY  annsal;
```

```
SQL> SELECT      ename, deptno, sal  
  2  FROM        emp  
  3  ORDER BY  2;
```

TOP Clause

- It is used to specify the number of records the query should return.
- Useful on large databases with thousands of records.

```
SELECT      [DISTINCT] { * | column [alias], ... }  
FROM        table  
[WHERE]      rownum <=10;
```

Summary

In this lesson, you should have learned how to:

- Use the WHERE clause to restrict rows of output
 - Use the comparison conditions
 - Use the BETWEEN, IN, LIKE, and NULL conditions
 - Apply the logical AND, OR, and NOT operators
- Use the ORDER BY clause to sort rows of output

```
SELECT      * | { [DISTINCT] column|expression [alias], ... }  
FROM        table  
[WHERE      condition(s)]  
[ORDER BY   {column, expr, alias} [ASC|DESC]];
```



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



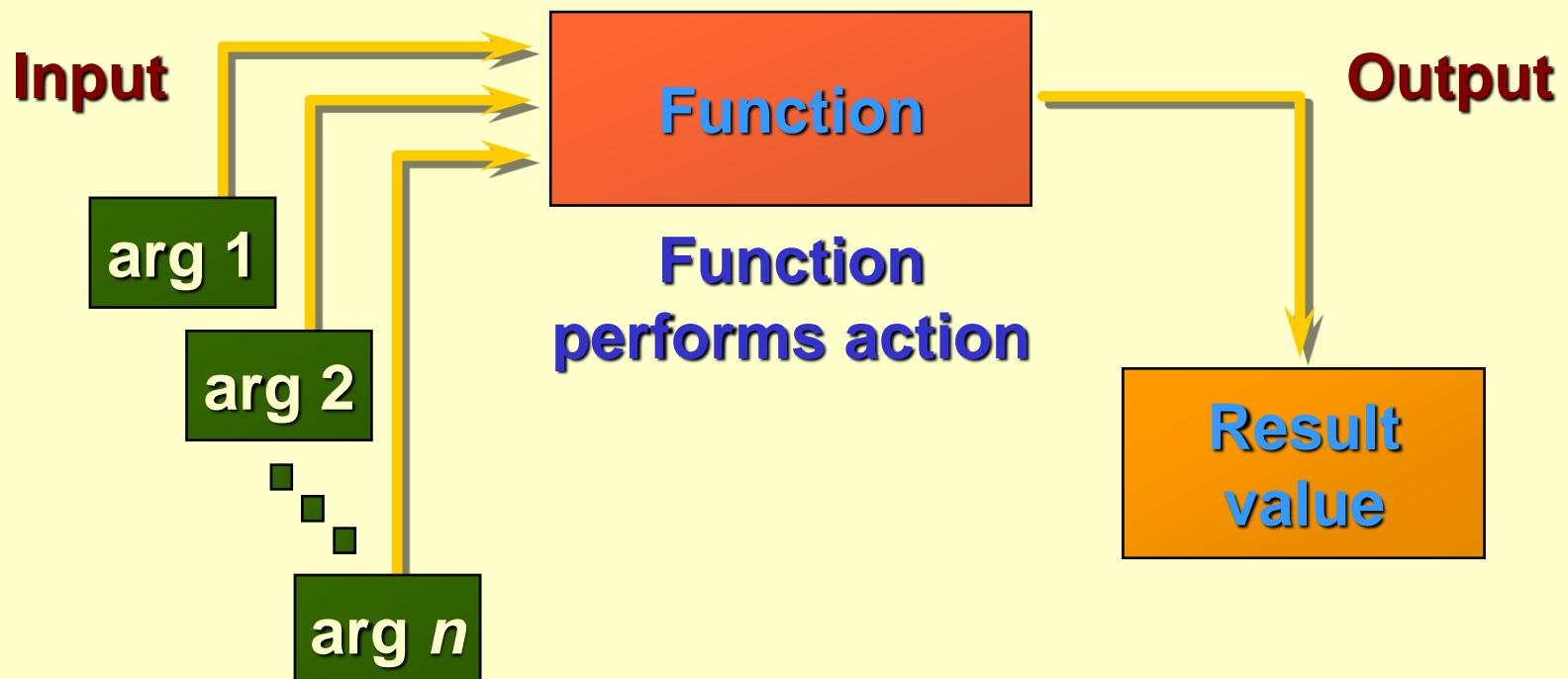
(SQL)

Single-Row Functions

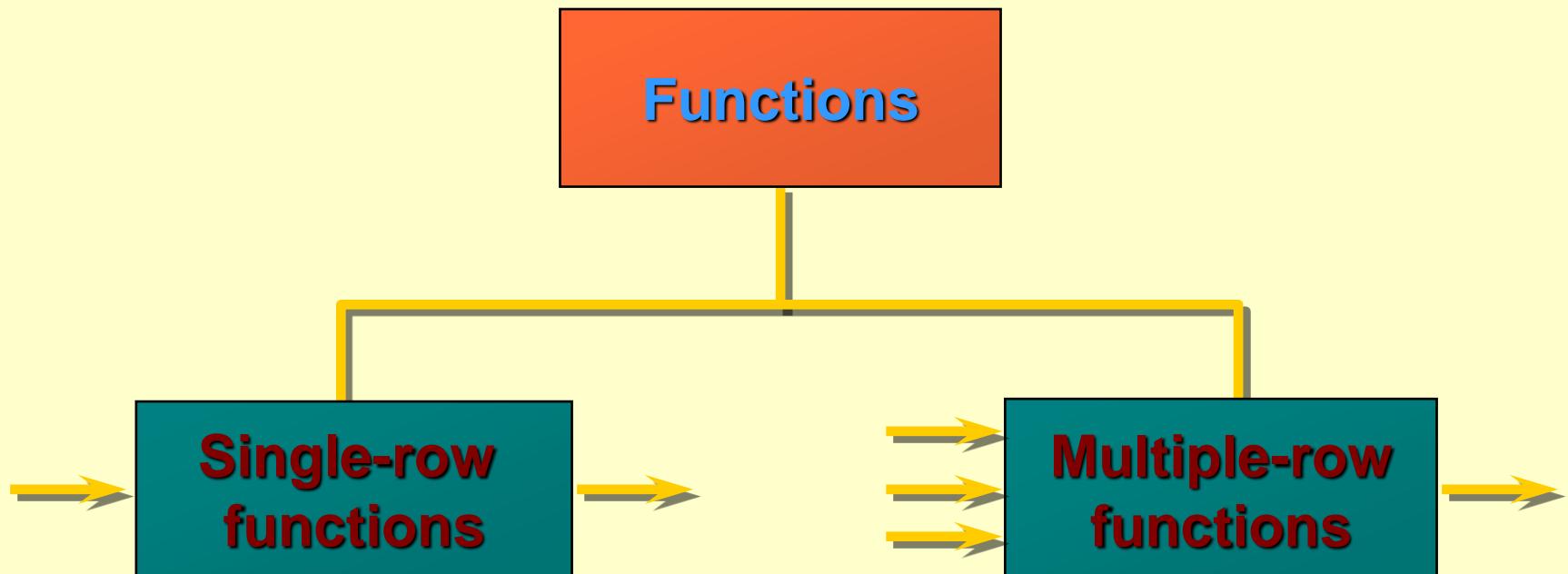
Asif Sohail

**University of the Punjab
Punjab University College of Information Technology (PUCIT)**

SQL Functions



Two Types of SQL Functions

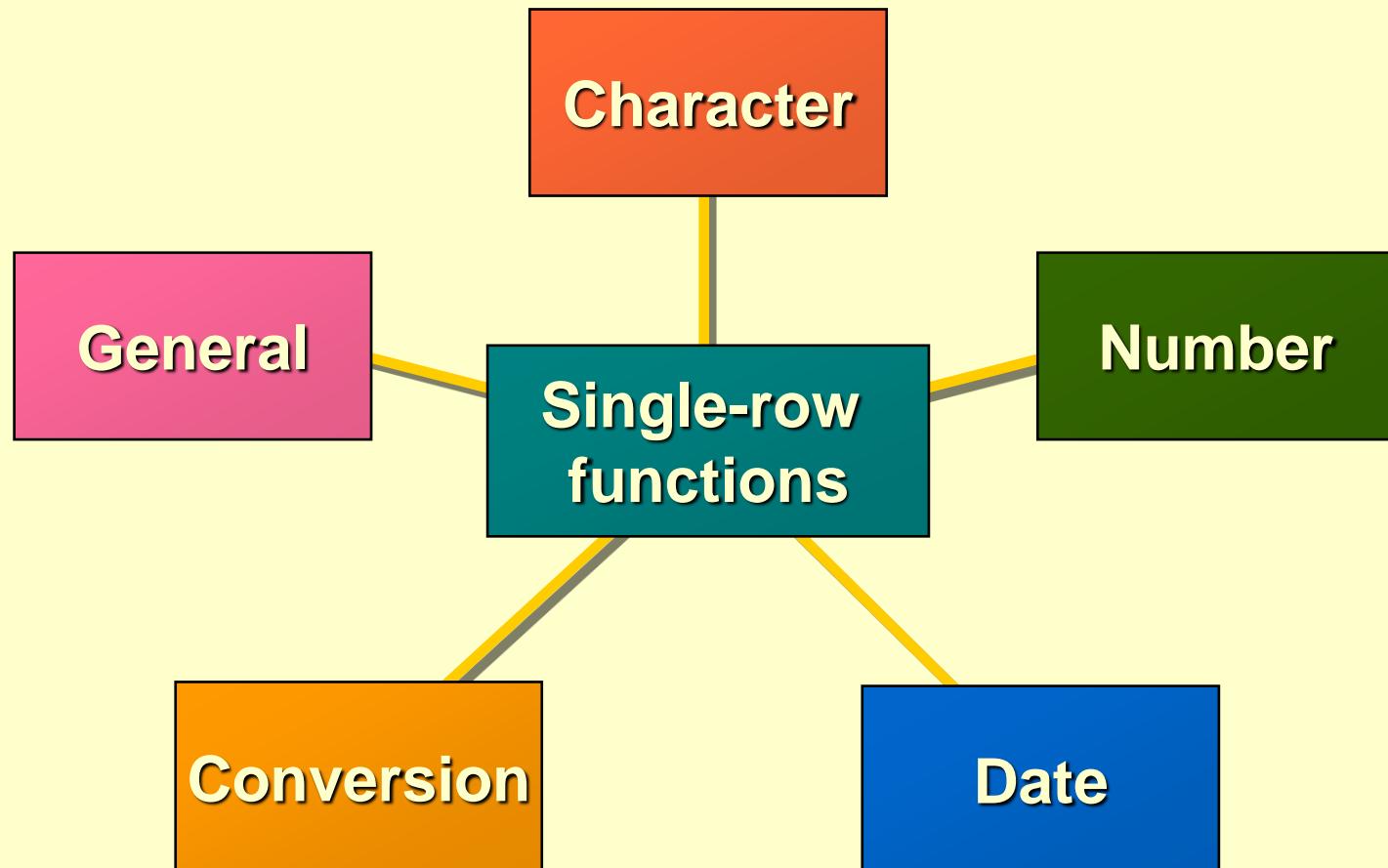


Single-Row Functions

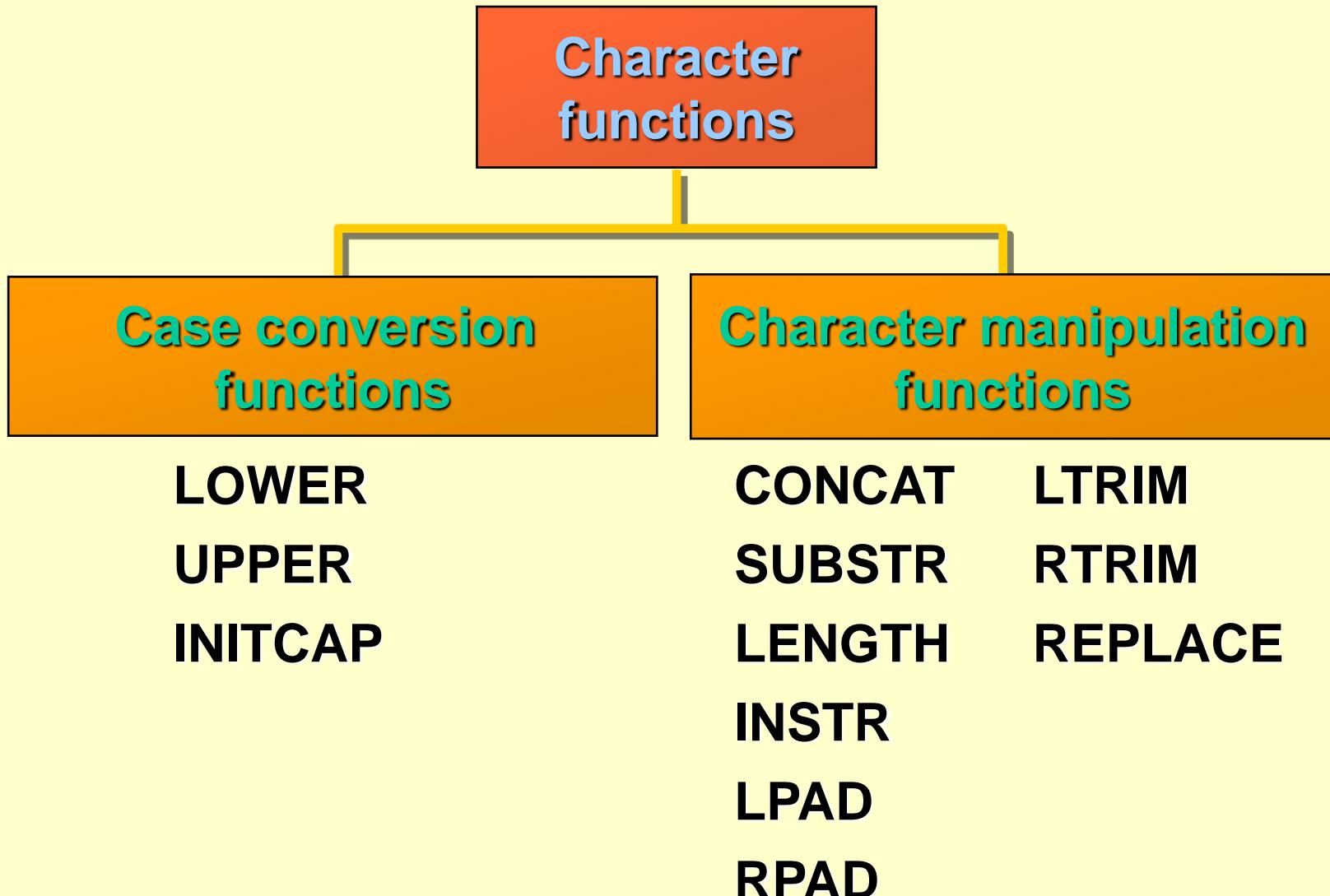
- Manipulate data items
- Accept arguments and return one value
- Act on each row returned
- Return one result per row
- May modify the datatype
- Can be nested

```
function_name (column|expression, [arg1, arg2, ...])
```

Single-Row Functions



Character Functions



Case Conversion Functions

- Convert case for character strings

Function	Result
<code>LOWER('SQL Course')</code>	<code>sql course</code>
<code>UPPER('SQL Course')</code>	<code>SQL COURSE</code>
<code>INITCAP('SQL Course')</code>	<code>Sql Course</code>
<code>INITCAP('SQL course')</code>	<code>Sql Course</code>

Using Case Conversion Functions

- Display the employee number, name, and department number for employee Blake.

```
SQL> SELECT    empno, ename, deptno  
2   FROM      emp  
3  WHERE     ename = 'blake';  
no rows selected
```

```
SQL> SELECT    empno, ename, deptno  
2   FROM      emp  
3  WHERE     LOWER(ename) = 'blake';
```

EMPNO	ENAME	DEPTNO
7698	BLAKE	30

Character Manipulation Functions

Function Syntax	Example	Result
SUBSTR(Str, start-pos, [length])	SUBSTR('Pakistan',2,3) SUBSTR('Pakistan',5) SUBSTR('Pakistan',-3)	aki stan tan
LPAD (Str, Padded-length, [Padded-char])	LPAD ('Pak", 10, '') LPAD ('Pak", 10)	*****Pak Pak
RPAD (Str, Padded-length, [Padded-char])	RPAD ('Pak", 10, '')	Pak*****
LTRIM (Str, [trim-str])	LTRIM ('***Pak','*)	Pak
RTRIM (Str, [trim-str])	RTRIM ('Pak***','*')	Pak
TRIM ([leading trailing[trim-str] from trim-str])	TRIM ('*' from '***Pak***')	Pak
INSTR(Str1, Str2, [start-pos], [Nth-appearance])	INSTR('Pakistan','a') INSTR('Pakistan','a',1,2)	2 7

Character Manipulation Functions

Function	Result
CONCAT('Fall', '2020')	Fall2020
SUBSTR('PUCIT',1,3)	PUC
SUBSTR('PUCIT',3)	CIT
SUBSTR('PUCIT',-4,2)	UC
LENGTH('PUCIT')	5
INSTR('PUCIT', 'C')	3
LPAD(sal,10,'*')	*****5000
LTRIM('***ABC',*)	ABC
RTRIM('ABC***',*)	ABC
REPLACE('ARIF','R','S')	ASIF

Using the Character Manipulation Functions

```
SQL> SELECT ename, CONCAT (ename, job), LENGTH(ename),  
2      INSTR(ename, 'A')  
3  FROM emp  
4 WHERE SUBSTR(job,1,5) = 'SALES';
```

ENAME	CONCAT (ENAME, JOB)	LENGTH (ENAME)	INSTR (ENAME, 'A')
MARTIN	MARTINSALESMAN	6	2
ALLEN	ALLENSALESMAN	5	1
TURNER	TURNERSALESMAN	6	0
WARD	WARDSALESMAN	4	2

Number Functions

- ROUND: Rounds value to specified decimal
 $\text{ROUND}(45.926, 2)$  45.93
- TRUNC: Truncates value to specified decimal
 $\text{TRUNC}(45.926, 2)$  45.92
- MOD: Returns remainder of division
 $\text{MOD}(1600, 300)$  100
- SQRT: Returns Square root of a number.
- POWER(A,B): Returns A raised to power B.
- ABS(A): Returns Absolute value of A.
- CEIL(A): Returns Smallest integer $\geq A$.
- FLOOR(A): Returns Largest integer $\leq A$.
- SIGN(A): Returns -1 or 0 or 1

Using the ROUND Function

```
SQL> SELECT ROUND(45.923,2), ROUND(45.923,0),
2      ROUND(45.923,-1)
3  FROM    DUAL;
```

ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
-----	-----	-----
45.92	46	50

Using the TRUNC Function

```
SQL> SELECT TRUNC(45.923,2), TRUNC(45.923),
2      TRUNC(45.923,-1)
3  FROM DUAL;
```

TRUNC(45.923,2)	TRUNC(45.923)	TRUNC(45.923,-1)	
-----	-----	-----	
45.92	45	40	

Using the MOD Function

- Calculate the remainder of the ratio of salary to commission for all employees whose job title is salesman.

```
SQL> SELECT    ename, sal, comm, MOD(sal, comm)  
  2  FROM      emp  
  3  WHERE     job = 'SALESMAN' ;
```

ENAME	SAL	COMM	MOD (SAL , COMM)
MARTIN	1250	1400	1250
ALLEN	1600	300	100
TURNER	1500	0	1500
WARD	1250	500	250

Number Functions

SQL> SELECT	CEIL(4.5) from dual;	Output: 5
SQL> SELECT	CEIL(-4.5) from dual;	Output: -4
SQL> SELECT	FLOOR(4.5) from dual;	Output: 4
SQL> SELECT	FLOOR(-4.5) from dual;	Output: -5
SQL> SELECT	SIGN(-4) from dual;	Output: -1
SQL> SELECT	SIGN(4) from dual;	Output: 1
SQL> SELECT	SIGN(0) from dual;	Output: 0

Other Number Functions:

EXP(A), LOG(A), LN(A),
SIN(A), COS(A), TAN(A),
SINH(A), COSH(A), TANH(A)

Working with Dates

- Oracle stores dates in an internal numeric format: century, year, month, day, hours, minutes, seconds.
- The default date format is mm/dd/yyyy OR DD-MON-YY.
- SYSDATE is a function returning date and time.
- DUAL is a dummy table used to view SYSDATE.

Arithmetic with Dates

- Add or subtract a number to or from a date for a resultant *date* value.
- Subtract two dates to find the *number* of days between those dates.
- Add *hours* to a date by dividing the number of hours by 24.

Using Arithmetic Operators with Dates

```
SQL> SELECT ename, (SYSDATE-hiredate)/7 WEEKS  
2  FROM emp  
3 WHERE deptno = 10;
```

ENAME	WEEKS
KING	830.93709
CLARK	853.93709
MILLER	821.36566

Date Functions

Function	Description
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

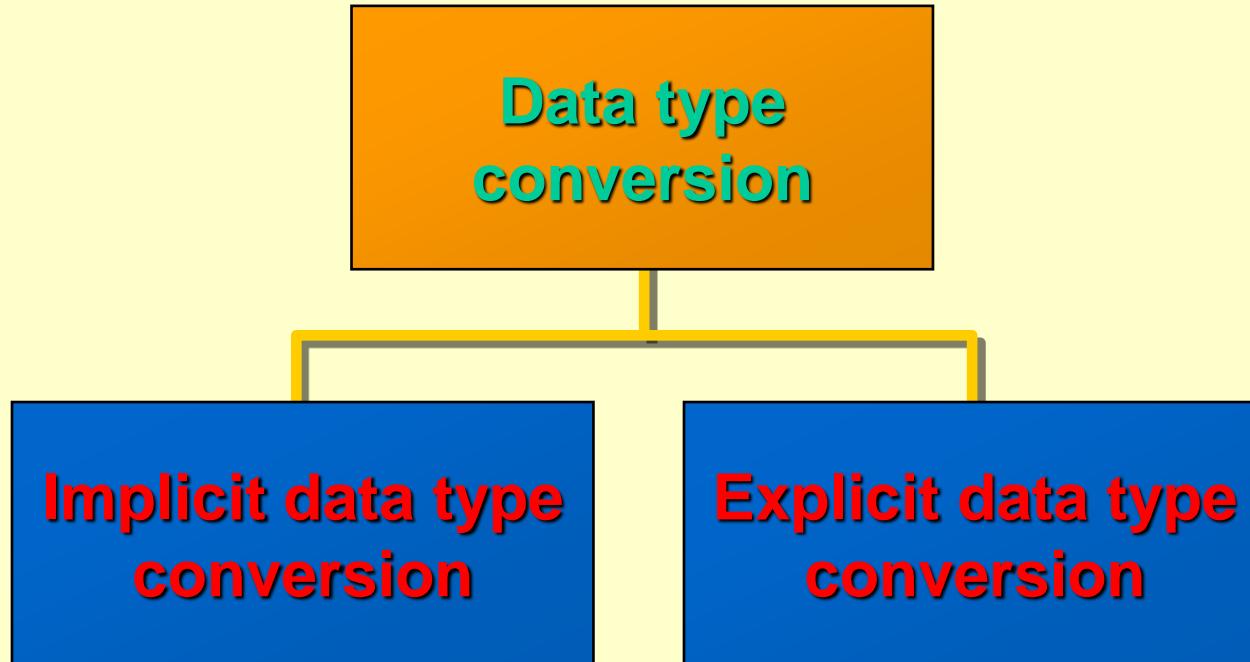
Using Date Functions

- **MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')**  **19.6774194**
- **ADD_MONTHS ('11-JAN-94',6)**  **'11-JUL-94'**
- **NEXT_DAY ('01-SEP-95','FRIDAY')**  **'08-SEP-95'**
- **LAST_DAY('01-SEP-95')**  **'30-SEP-95'**

Using Date Functions

- **ROUND('25-JUL-95','MONTH')**  **01-AUG-95**
- **ROUND('25-JUL-95','YEAR')**  **01-JAN-96**
- **TRUNC('25-JUL-95','MONTH')**  **01-JUL-95**
- **TRUNC('25-JUL-95','YEAR')**  **01-JAN-95**

Conversion Functions

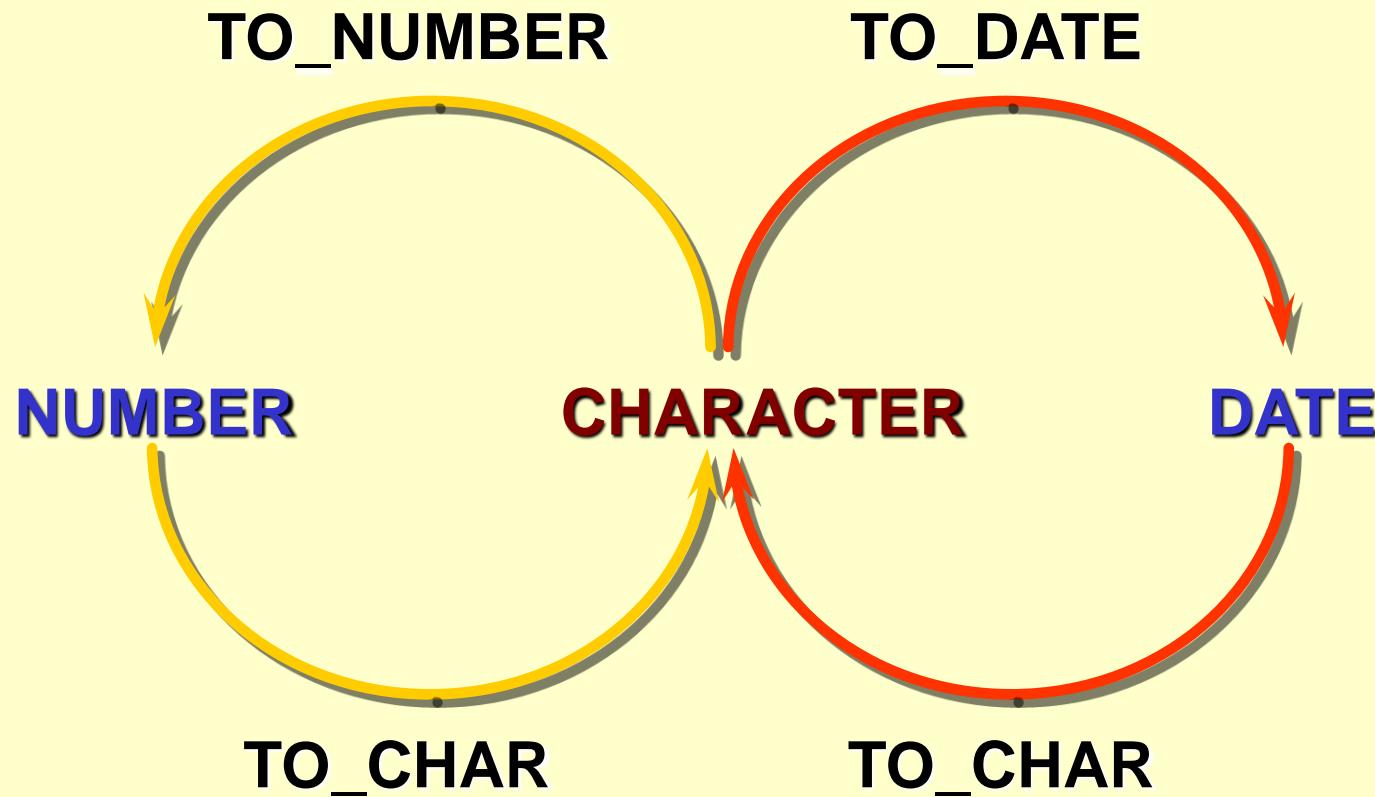


Implicit Data type Conversion

- For assignments, the Oracle can automatically convert the following:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

Explicit Data type Conversion



TO_CHAR Function with Dates

```
TO_CHAR(date, 'fmt')
```

YYYY	Full year in numbers
YEAR	Year spelled out
MM	Two-digit value for month
MONTH	Full name of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day

Using TO_CHAR Function with Dates

```
SQL> SELECT ename,  
2          TO_CHAR(hiredate, 'fmDD Month YYYY') HIREDATE  
3  FROM    emp;
```

ENAME	HIREDATE
KING	17 November 1981
BLAKE	1 May 1981
CLARK	9 June 1981
JONES	2 April 1981
MARTIN	28 September 1981
ALLEN	20 February 1981
...	
14 rows selected.	

TO_CHAR Function with Numbers

```
TO_CHAR(number, 'fmt')
```

- Use these formats with the TO_CHAR function to display a number value as a character:

9	Represents a number
0	Forces a zero to be displayed
\$	Places a floating dollar sign
.	Prints a decimal point
,	Prints a thousand indicator

Using TO_CHAR Function with Numbers

```
SQL> SELECT      TO_CHAR(sal, '$99,999')  SALARY  
2   FROM        emp  
3   WHERE       ename = 'SCOTT' ;
```

SALARY

\$3,000

TO_NUMBER and TO_DATE Functions

- Convert a character string to a number format using the **TO_NUMBER** function

```
TO_NUMBER(char[, 'fmt'])
```

- Convert a character string to a date format using the **TO_DATE** function

```
TO_DATE(char[, 'fmt'])
```

```
SQL> SELECT * from emp
2 WHERE hiredate = TO_DATE('Oct 20,09', 'mon dd,yy');
```

General Functions

These functions work with any data type and pertain to using nulls.

- NVL (expr1, expr2)
- NVL2 (expr1, expr2, expr3)
- NULLIF (expr1, expr2)

NVL Function

- Converts a null to some other value.
- Data types that can be used are date, character, and number.
- Data types must match:
 - `NVL (commission_pct, 0)`
 - `NVL (hire_date, '01-JAN-97')`
 - `NVL (job_id, 'No Job Yet')`
- `//return expression 2 when expr1 is null otherwise return expr1`

NVL2 Function

- It provides alternate values both for NULL and NOT NULL cases.
- NVL2 (expr1, expr2 NOT NULL case, expr3 NULL case)
- NVL2(comm, sal+comm, sal)

```
//return expr3 when expr1 is null,  
else return expr2
```

NVLIF Function

- `NVLIF (expr1, expr2)`
- It returns NULL if both expr1 and expr2 are same.
- `select name, job
NVLIF(LENGTH(ename), LENGTH(job))
result from emp;`
//return expr1 when expr1!=expr2 otherwise return
NULL

Conditional Expressions

- Provide the use of IF-THEN-ELSE logic within a SQL statement
- Use two methods:
 - DECODE function
 - CASE expression

DECODE Function

- Facilitates conditional inquiries by doing the work of a **CASE** or **IF-THEN-ELSE** statement

```
DECODE(col/expression, search1, result1
       [, search2, result2, . . . ,]
       [, default])
```

Using the DECODE Function

```
SQL> SELECT job, sal,
2      DECODE(job, 'ANALYST', SAL*1.1,
3              'CLERK',     SAL*1.15,
4              'MANAGER',   SAL*1.20,
5              SAL)
6      REVISED_SALARY
7  FROM emp;
```

JOB	SAL	REVISED_SALARY
PRESIDENT	5000	5000
MANAGER	2850	3420
MANAGER	2450	2940
...		
14 rows selected.		

The CASE Expression

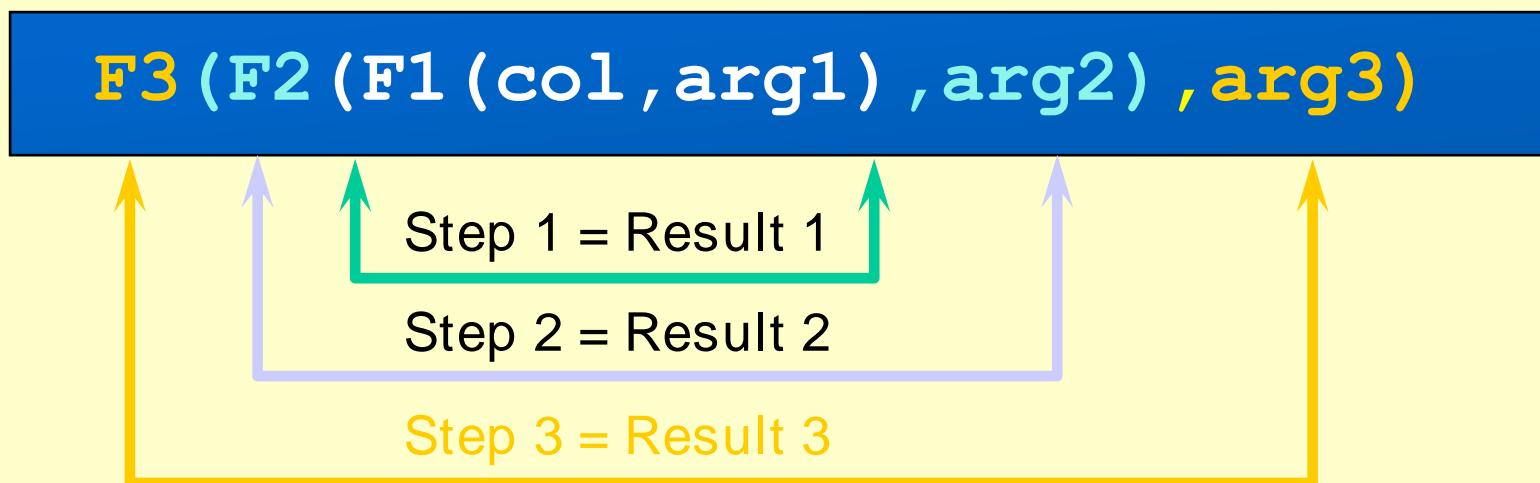
- Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
            [WHEN comparison_expr2 THEN return_expr2
             WHEN comparison_exprn THEN return_exprn
             ELSE else_expr]
END
```

```
SELECT ename, job, sal,
       CASE job      WHEN 'CLERK'    THEN 1.10*sal
                     WHEN 'SALESMAN'  THEN 1.15*sal
                     WHEN 'MANAGER'   THEN 1.20*sal
                     ELSE          sal END      "REVISED_SALARY"
FROM emp;
```

Nesting Functions

- Single-row functions can be nested to any level.
- Nested functions are evaluated from deepest level to the least-deep level.



Nesting Functions

```
SQL> SELECT    ename,  
2          NVL(TO_CHAR(mgr) , 'No Manager')  
3      FROM      emp  
4      WHERE     mgr  IS  NULL;
```

ENAME	NVL (TO_CHAR (MGR) , 'NOMANAGER')
KING	No Manager



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



(SQL)

Aggregating Data Using Group Functions

Asif Sohail

**University of the Punjab
Punjab University College of Information Technology (PUCIT)**

What Are Group Functions?

- Group functions operate on sets of rows to give one result per group.

EMP

DEPTNO	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250
30	950
30	1500
30	1250

“maximum
salary in
the EMP table”

MAX (SAL)
----- 5000

Types of Group Functions

- MAX
- MIN
- SUM
- AVG
- COUNT

Using Group Functions

```
SELECT      [column,] group_function(column)
FROM        table
[WHERE       condition]
[GROUP BY    column]
[ORDER BY    column] ;
```

Using MIN and MAX Functions

- You can use MIN and MAX for any datatype.

```
SQL> SELECT    MIN(hiredate) , MAX(hiredate)  
2   FROM        emp;
```

MIN (HIRED)	MAX (HIRED)
-----	-----
17-DEC-80	12-JAN-83

Using AVG and SUM Functions

- You can use AVG and SUM for numeric data.

```
SQL> SELECT    AVG(sal), MAX(sal),  
2          MIN(sal), SUM(sal)  
3  FROM      emp  
4  WHERE     job LIKE 'SALES%' ;
```

AVG (SAL)	MAX (SAL)	MIN (SAL)	SUM (SAL)
1400	1600	1250	5600

Using the COUNT Function

- COUNT(*) returns the number of rows in a table satisfying a given condition (if any).

```
SQL> SELECT COUNT (*)
  2  FROM emp
  3  WHERE deptno = 30;
```

COUNT (*)

6

Using the COUNT Function

- COUNT(expr) returns the number of rows with non-null values for the $expr$.

```
SQL> SELECT      COUNT (comm)
      2  FROM        emp
      3  WHERE       deptno = 30;
```

COUNT (COMM)

4

Using the COUNT Function

- COUNT (DISTINCT expr) returns the number of distinct non-null values of the *expr*.

```
SQL> SELECT      COUNT(DISTINCT deptno)
  2  FROM        emp;
```

```
COUNT (DISTINCT deptno)
-----
4
```

Group Functions and Null Values

- Group functions ignore null values in the column.

```
SQL> SELECT AVG(comm)
  2  FROM emp;
```

AVG (COMM)

550

Using the NVL Function with Group Functions

- The NVL function forces group functions to include null values.

```
SQL> SELECT AVG(NVL(comm, 0))  
2   FROM emp;
```

```
AVG (NVL (COMM, 0))  
-----  
157.14286
```

Creating Groups of Data

EMP

DEPTNO	SAL
10	2450
	5000
	1300
20	800
	1100
	3000
	3000
	2975
	1600
30	2850
	1250
	950
	1500
	1250

2916.6667

2175

“average
salary
in EMP
table
for each
department”

1566.6667

DEPTNO	AVG (SAL)
10	2916.6667
20	2175
30	1566.6667

Creating Groups of Data: GROUP BY Clause

```
SELECT      column, group_function(column)
FROM        table
[WHERE       condition]
[GROUP BY   group_by_expression]
[ORDER BY   column] ;
```

- Divide rows in a table into smaller groups by using the GROUP BY clause.

Using the GROUP BY Clause

- All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SQL> SELECT      deptno,  AVG(sal)
  2  FROM        emp
  3  GROUP BY  deptno;
```

DEPTNO	AVG (SAL)
10	2916.6667
20	2175
30	1566.6667

Using the GROUP BY Clause

- The GROUP BY column does not have to be in the SELECT list.

```
SQL> SELECT      AVG(sal)
  2  FROM        emp
  3  GROUP BY deptno;
```

AVG(SAL)

2916.6667
2175
1566.6667

Grouping by More Than One Column

EMP

DEPTNO	JOB	SAL
10	MANAGER	2450
	PRESIDENT	5000
	CLERK	1300
20	CLERK	800
	CLERK	1100
	ANALYST	3000
	ANALYST	3000
	MANAGER	2975
30	SALESMAN	1600
	MANAGER	2850
	SALESMAN	1250
	CLERK	950
	SALESMAN	1500
	SALESMAN	1250

“sum salaries in the EMP table for each job, grouped by department”

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
	MANAGER	2450
	PRESIDENT	5000
20	ANALYST	6000
	CLERK	1900
	MANAGER	2975
	CLERK	950
30	MANAGER	2850
	SALESMAN	5600
	SALESMAN	1250

Using the GROUP BY Clause on Multiple Columns

```
SQL> SELECT      deptno, job, sum(sal)
  2  FROM        emp
  3  GROUP BY    deptno, job;
```

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	CLERK	1900
...		
9 rows selected.		

Illegal Queries Using Group Functions

- Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause.

```
SQL> SELECT deptno, COUNT(ename)
  2  FROM emp;
```

Column missing in the GROUP BY clause

```
SELECT deptno, COUNT(ename)
      *
ERROR at line 1:
ORA-00937: not a single-group group function
```

Illegal Queries Using Group Functions

- You cannot use the WHERE clause to restrict groups.
- You use the HAVING clause to restrict groups.

```
SQL> SELECT      deptno,  AVG(sal)
  2  FROM        emp
  3  WHERE       AVG(sal) > 2000
  4  GROUP BY    deptno;
```

```
WHERE AVG(sal) > 2000
*
```

```
ERROR at line 3
```

```
ORA-00934: group function is not allowed here
```

*Cannot use the WHERE clause
to restrict groups*

Excluding Group Results

EMP

DEPTNO	SAL
10	2450
	5000
	1300
20	800
	1100
	3000
	3000
	2975
30	1600
	2850
	1250
	950
	1500
	1250

5000

3000

2850

**“maximum
salary
per department
greater than
\$2900”**

DEPTNO	MAX (SAL)
10	5000
20	3000

Excluding Group Results: HAVING Clause

- Use the HAVING clause to restrict groups
 - Rows are grouped.
 - The group function is applied.
 - Groups matching the HAVING clause are displayed.

```
SELECT      column, group_function
FROM        table
[WHERE       condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column] ;
```

Using the HAVING Clause

```
SQL> SELECT      deptno,  max(sal)
  2  FROM        emp
  3  GROUP BY    deptno
  4  HAVING      max(sal)>2900;
```

DEPTNO	MAX (SAL)
-----	-----
10	5000
20	3000

Using the HAVING Clause

```
SQL> SELECT      job, SUM(sal) PAYROLL  
  2  FROM        emp  
  3  WHERE       job NOT LIKE 'SALES%'  
  4  GROUP BY    job  
  5  HAVING      SUM(sal)>5000  
  6  ORDER BY    SUM(sal);
```

JOB	PAYROLL
ANALYST	6000
MANAGER	8275

Nesting Group Functions

- Display the maximum average salary.

```
SQL> SELECT      max (avg (sal))  
  2  FROM        emp  
  3  GROUP BY    deptno;
```

MAX (AVG (SAL))

2916.6667

Summary

```
SELECT      column, group_function(column)
FROM        table
[WHERE       condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column] ;
```

- Order of evaluation of the clauses:
 - WHERE clause
 - GROUP BY clause
 - HAVING clause



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



(SQL)

Displaying Data from Multiple Tables

Asif Sohail

**University of the Punjab
Punjab University College of Information Technology (PUCIT)**

Objectives

After completing this lesson, you should be able to do the following:

- Write SELECT statements to access data from more than one table using equality and non-equality joins
- View data that generally does not meet a join condition by using outer joins
- Join a table to itself by using a self join

Obtaining Data from Multiple Tables

EMP

EMPNO	ENAME	...	DEPTNO
7839	KING	...	10
7698	BLAKE	...	30
...			
7934	MILLER	...	10

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

EMPNO DEPTNO LOC

7839	10	NEW YORK
7698	30	CHICAGO
7782	10	NEW YORK
7566	20	DALLAS
7654	30	CHICAGO
7499	30	CHICAGO
...		

14 rows selected.

What Is a Join?

- Use a join to query data from more than one table.

```
SELECT      table1.column, table2.column  
FROM        table1, table2  
WHERE       table1.column1 = table2.column2;
```

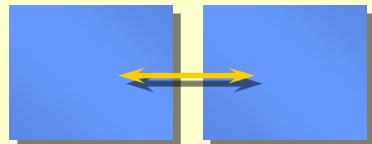
- Write the join condition in the WHERE clause.
- Prefix the column name with the table name when the same column name appears in more than one table.
- An alternate syntax for join operation introduced with oracle 9i is:

```
SELECT      table1.column, table2.column  
FROM        table1 {join type} JOIN table2  
[ON          table1.column1 = table2.column2];
```

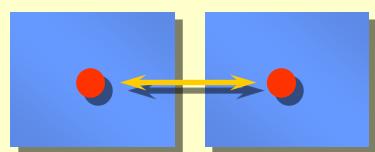
- Join type can be INNER (EQUI), LEFT, RIGHT, FULL OUTER...

Types of Joins

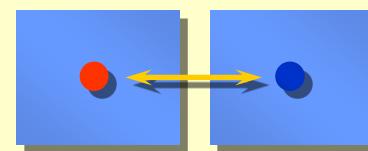
Cartesian join



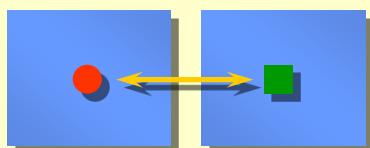
Equijoin



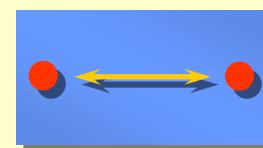
Outer join

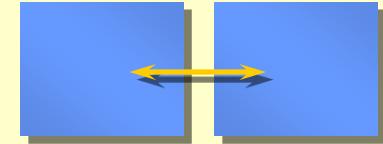


Non-equijoin



Self join





1. Cross Join or Cartesian Product

- A Cartesian product is formed when:
 - A join condition is omitted
 - A join condition is invalid
- Select ename, dname from emp,dept; OR
- Select ename, dname from emp **cross join** dept;
- All rows in the first table are joined to all rows in the second table
- The number of rows returned by cross join is the product of the number of rows in each table.
- To avoid a Cartesian product, always include a valid join condition in a WHERE clause.

Generating a Cartesian Product

EMP (14 rows)

EMPNO	ENAME	...	DEPTNO
7839	KING	...	10
7698	BLAKE	...	30
...			
7934	MILLER	...	10

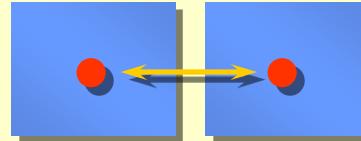
DEPT (4 rows)

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

**“Cartesian
product:
 $14 \times 4 = 56$ rows”**

ENAME	DNAME
KING	ACCOUNTING
BLAKE	ACCOUNTING
...	
KING	RESEARCH
BLAKE	RESEARCH
...	
56 rows selected.	

2. Equijoin or Inner Join



EMP

EMPNO	ENAME	DEPTNO
7839	KING	10
7698	BLAKE	30
7782	CLARK	10
7566	JONES	20
7654	MARTIN	30
7499	ALLEN	30
7844	TURNER	30
7900	JAMES	30
7521	WARD	30
7902	FORD	20
7369	SMITH	20
...		
14 rows selected.		

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
30	SALES	CHICAGO
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
20	RESEARCH	DALLAS
20	RESEARCH	DALLAS
...		
14 rows selected.		

Foreign key

Primary key

Euijoin / Inner Join

- Inner joins are the most commonly used joins. For this reason, when people refer simply to a "join," they most likely mean an "inner join."
- The pre-requisite for inner join is that its participating tables must have at least one common attribute between them.

```
SELECT col-1, col-2, ....
```

```
FROM table-1, table-2, ...
```

```
WHERE table-1.Join_Col = table-2. Join_Col ...
```

```
SELECT col-1, col-2, ....
```

```
FROM table-1 [INNER | EQUI] JOIN table-2, ...
```

```
{ON table-1. Join_Col = table-2. Join_Col ... |USING (Join_Col)}
```

```
[WHERE clause]
```

Euijoin / Inner Join

- Select ename, dname from emp,dept
where emp.deptno = dept.deptno;
- Select ename, dptno, dname from emp inner join dept
on emp.deptno = dept.deptno;
- Select ename, dname from emp join dept
on emp.deptno = dept.deptno;
- Select ename, dname from emp join dept
using (deptno);
- Select e.empno, e.ename, d.deptno, d.dname
from emp e join dept d
on e.deptno = d.deptno;

Retrieving Records with Equijoin

```
SELECT      emp.empno, emp.ename, emp.deptno,  
            dept.deptno, dept.loc  
FROM        emp, dept  
WHERE       emp.deptno = dept.deptno;
```

EMPNO	ENAME	DEPTNO	DEPTNO	LOC
-----	-----	-----	-----	-----
7839	KING	10	10	NEW YORK
7698	BLAKE	30	30	CHICAGO
7782	CLARK	10	10	NEW YORK
7566	JONES	20	20	DALLAS
...				
14 rows selected.				

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Improve performance by using table prefixes.
- Distinguish columns that have identical names but reside in different tables by using column aliases.

Additional Search Conditions Using the AND Operator

```
SELECT      emp.empno, emp.ename, emp.deptno,  
            dept.deptno, dept.loc  
FROM        emp, dept  
WHERE       emp.deptno=dept.deptno  
           AND dept.dname like 'SALES' ;
```

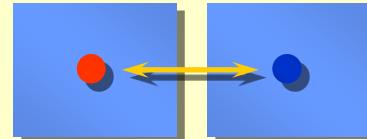
Using Table Aliases

- Simplify queries by using table aliases.

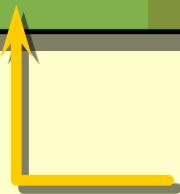
```
SQL> SELECT emp.empno, emp.ename, emp.deptno,  
2      dept.deptno, dept.loc  
3  FROM  emp, dept  
4 WHERE emp.deptno=dept.deptno;
```

```
SQL> SELECT e.empno, e.ename, e.deptno,  
2      d.deptno, d.loc  
3  FROM  emp e, dept d  
4 WHERE e.deptno=d.deptno;
```

3. Outer Joins



EMP		DEPT	
ENAME	DEPTNO	DEPTNO	DNAME
KING	10	10	ACCOUNTING
BLAKE	30	30	SALES
CLARK	10	10	ACCOUNTING
JONES	20	20	RESEARCH
...		...	
		40	OPERATIONS



**No employee in the
OPERATIONS department**

Outer Joins

- Sometimes, while performing a join between two tables, you need to return all the rows from one table even when there are no corresponding rows in the other table.
- The inner join returns only those rows from the two tables that satisfy the join condition. Thus equijoin will not return the departments that don't have any employee or employees whose department is NULL.

FROM *table1* { LEFT | RIGHT | FULL } [OUTER]

JOIN *table2*

ON *table-1*. Join_Col = *table-2*. Join_Col

Outer Joins

- **LEFT**

Specifies that the results be generated using all rows from *table1*. For those rows in *table1* that don't have corresponding rows in *table2*, NULLs are returned in the result set for the *table2* columns.

- **RIGHT**

Specifies that the results be generated using all rows from *table2*. For those rows in *table2* that don't have corresponding rows in *table1*, NULLs are returned in the result set for the *table1* columns.

- **FULL**

Specifies that the results be generated using all rows from *table1* and *table2*. For those rows in *table1* that don't have corresponding rows in *table2*, NULLs are returned in the result set for the *table2* columns and vice-versa.

- **OUTER**

Specifies that you are performing an OUTER join. This keyword is optional. If you use LEFT, RIGHT, or FULL, Oracle automatically assumes an outer join. The OUTER keyword is for completeness' sake, and complements the INNER keyword.

Outer Joins

- You use an outer join to also see rows that do not usually meet the join condition.
- Outer join operator is the plus sign (**(+)**).
- Left Outer Join.

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column = table2.column (+);
```

- Right Outer Join.

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column (+) = table2.column ;
```

Left Outer Joins

To list any employees not assigned to a particular department.

```
SQL> SELECT e.ename, d.deptno, d.dname  
2 FROM emp e, dept d  
3 WHERE e.deptno = d.deptno(+)
```

```
SQL> SELECT e.ename, d.deptno, d.dname  
2 FROM emp e LEFT JOIN dept d  
3 ON e.deptno = d.deptno
```

ENAME	DEPTNO	DNAME
KING	10	ACCOUNTING
CLARK	10	ACCOUNTING
...		
BABU KHAN		

15 rows selected.

Right Outer Joins

To retrieve the departments that don't have any employees.

```
SQL> SELECT e.ename, d.deptno, d.dname  
2 FROM emp e, dept d  
3 WHERE e.deptno(+) = d.deptno
```

```
SQL> SELECT e.ename, d.deptno, d.dname  
2 FROM emp e RIGHT OUTER JOIN dept d  
3 ON e.deptno = d.deptno
```

ENAME	DEPTNO	DNAME
KING	10	ACCOUNTING
CLARK	10	ACCOUNTING
...		
	40	OPERATIONS
15 rows selected.		

Illegal Queries Using Outer Joins

```
SQL> SELECT      e.ename, d.deptno, d.dname
  2  FROM        emp e, dept d
  3  WHERE      e.deptno(+) = d.deptno(+)
  4  ORDER BY e.deptno;
```

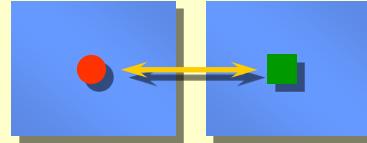
Cannot use left and right outer
join simultaneously.

Using Left and Right Outer Join in One Query

```
SQL> SELECT      e.ename, d.deptno, d.dname
  2  FROM        emp e, dept d
  3  WHERE      e.deptno(+) = d.deptno
  4  UNION
  5  SELECT      e.ename, d.deptno, d.dname
  6  FROM        emp e, dept d
  3  WHERE      e.deptno = d.deptno(+)
```

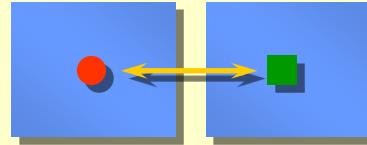
- Similarly, We can use **INTERSECT** and **MINUS** to combine two or more select statements into a single query, provided that the select-list of all the queries remain the same.

```
SQL> SELECT      e.ename, d.deptno, d.dname
  2  FROM        emp e FULL OUTER JOIN dept d
  3  ON          e.deptno = d.deptno
```



4. Non-Equiijoins

- The join condition determines whether a join is an equijoin or a non-equijoin.
- When a join condition relates two tables by equating the columns from the tables, it is an equijoin.
- When a join condition relates two tables by an operator other than equality, it is a non-equijoin.
- Non-equijoin is used to retrieve data from two tables that don't have a common attribute between them, but still the tables are related with each other.



4. Non-Equiijoins

EMP

EMPNO	ENAME	SAL
7839	KING	5000
7698	BLAKE	2850
7782	CLARK	2450
7566	JONES	2975
7654	MARTIN	1250
7499	ALLEN	1600
7844	TURNER	1500
7900	JAMES	950
...		
14 rows selected.		

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

**“salary in the EMP
table is between
low salary and high
salary in the SALGRADE
table”**

Retrieving Records with Non-Equiijoins

```
SQL> SELECT e.ename, e.sal, s.grade  
2  FROM emp e, salgrade s  
3  WHERE e.sal  
4  BETWEEN s.losal AND s.hisal;
```

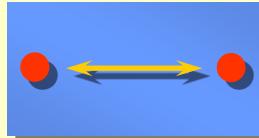
```
SQL> SELECT e.ename, e.sal, s.grade  
2  FROM emp e JOIN salgrade s  
3  ON e.sal  
4  BETWEEN s.losal AND s.hisal;
```

ENAME	SAL	GRADE
JAMES	950	1
SMITH	800	1
ADAMS	1100	1

...

14 rows selected.

5. Self Joins



EMP (WORKER)

EMPNO	ENAME	MGR
7839	KING	
7698	BLAKE	7839
7782	CLARK	7839
7566	JONES	7839
7654	MARTIN	7698
7499	ALLEN	7698

EMP (MANAGER)

EMPNO	ENAME
7839	KING
7839	KING
7839	KING
7698	BLAKE
7698	BLAKE

“MGR in the WORKER table is equal to EMPNO in the MANAGER table”

One row in a table is related to another row of the same table.

Joining a Table to Itself

```
SELECT e.ename "Employee", m.ename "Manager"  
FROM   emp, emp m  
WHERE  e.mgr = m.empno;
```

```
SELECT e.ename "Employee", m.ename "Manager"  
FROM   emp e JOIN emp m  
ON     e.mgr = m.empno;
```

Employee	Manager

BLAKE	KING
CLARK	KING
JONES	KING
MARTIN	BLAKE
...	

Joining a Table to Itself

```
SELECT worker.ename || ' works for ' || manager.ename  
FROM   emp worker, emp manager  
WHERE  worker.mgr = manager.empno;
```

```
SELECT worker.ename || ' works for ' || manager.ename  
FROM   emp worker JOIN emp manager  
ON     worker.mgr = manager.empno;
```

WORKER.ENAME || 'WORKSFOR' || MANAG

BLAKE works for KING
CLARK works for KING
JONES works for KING
MARTIN works for BLAKE
...

13 rows selected.

Self Outer Join

The last query doesn't return the employee, whose mgr is NULL. To include manager less employees, the following query is used:

```
SELECT e.ename "Employee", m.ename "Manager"  
FROM   emp e LEFT OUTER JOIN emp m  
ON     e.mgr = m.empno;
```

Employee	Manager
BLAKE	KING
CLARK	KING
JONES	KING
MARTIN	BLAKE
KING	
...	

14 rows selected.

Joining More Than Two Tables

CUSTOMER

NAME	CUSTID
JOCKSPORTS	100
TKB SPORT SHOP	101
VOLLYRITE	102
JUST TENNIS	103
K+T SPORTS	105
SHAPE UP	106
WOMENS SPORTS	107
...	...
9 rows selected.	

ORD

CUSTID	
101	
102	
104	
106	
102	
106	
106	
...	
21 rows selected.	

ORDID

ORDID
610
611
612
601
602
...

ITEM

ORDID	ITEMID
610	3
611	1
612	1
601	1
602	1
...	

64 rows selected.

- To join n tables together, you need a minimum of $n-1$ join conditions. For example, to join three tables, a minimum of two joins is required.

Joining More Than Two Tables

```
SELECT col-1, col-2, ....
```

```
FROM table-1, table-2, ..., table-n
```

```
WHERE table-1.Join_Col = table-2. Join_Col AND  
table-1.Join_Col = table-3. Join_Col AND ...
```

```
SELECT col-1, col-2, ....
```

```
FROM table-1 [INNER | EQUI | LEFT | RIGHT]
```

```
JOIN table-2 ON table-1.Join_Col = table-2. Join_Col
```

```
JOIN table-3 ON table-1.Join_Col = table-3. Join_Col
```

```
[WHERE clause]
```

Joining More Than Two Tables

```
SELECT NAME, C.CUSTID, O.ORDID, ITEMID  
FROM CUSTOMER C, ORD O, ITEM I  
WHERE C.CUSTID = C.CUSTID AND  
      O.ORDID = I.ORDID
```

```
SELECT NAME, C.CUSTID, O.ORDID, ITEMID  
FROM CUSTOMER C, ORD O, ITEM I  
JOIN ORD O ON C.CUSTID = C.CUSTID  
JOIN ITEM I ON O.ORDID = I.ORDID
```

Joining More Than Two Tables

```
SELECT NAME, C.CUSTID, O.ORDID, ITEMID  
FROM CUSTOMER C, ORD O, ITEM I  
WHERE C.CUSTID = C.CUSTID AND  
O.ORDID = I.ORDID
```

```
SELECT E.ENAME, D.DEPTNO, DNAME, GRADE  
FROM EMP E  
JOIN DEPT D ON E.DEPTNO = D.DEPTNO  
JOIN SALGRADE ON SAL BETWEEN LOSAL AND HISAL
```

Practice Questions

1. FIND THOSE EMPLOYEES WHO ARE WORKING IN ACCOUNTING DEPARTMENT.
2. FIND THOSE EMPLOYEES WHOSE DEPARTMENT LOCATION IS NEWYORK.
3. LIST THOSE DEPARTMENTS WHO DOES NOT HAVE ANY EMPLOYEES.
4. FIND THE MANAGER OF 'SCOTT'.



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



(SQL) Subqueries

Asif Sohail

**University of the Punjab
Punjab University College of Information Technology (PUCIT)**

Objectives

- After completing this lesson, you should be able to do the following:
 - Describe the types of problems that subqueries can solve
 - Define subqueries
 - List the types of subqueries
 - Write single-row and multiple-row subqueries

Using a Subquery to Solve a Problem

- “Who has a salary greater than Jones’?”

Main Query



“Which employees have a salary greater than Jones’ salary?”



Subquery



“What is Jones’ salary?”

Subqueries

- Query within a query is called subquery.

```
SELECT      select_list
FROM        table
WHERE       expr operator
            (SELECT      select_list
             FROM       table) ;
```

- The subquery (inner query) executes once before the main query.
- The result of the subquery is used by the main query (outer query).

Using a Subquery

```
SQL> SELECT ename  
  2  FROM emp  
  3 WHERE sal >  
  4  
  5  
  6
```

(SELECT sal
 FROM emp
 WHERE empno=7566) ;

2975

ENAME

KING

FORD

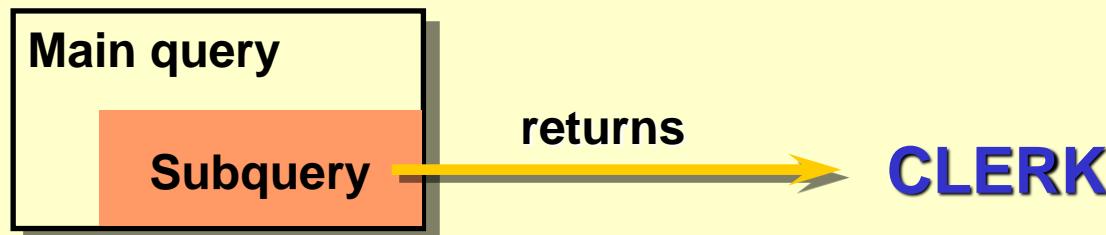
SCOTT

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison operator.
- ORDER BY clause is mostly not required in a subquery.
- Use single-row operators with single-row subqueries.
- Use multiple-row operators with multiple-row subqueries.

Types of Subqueries

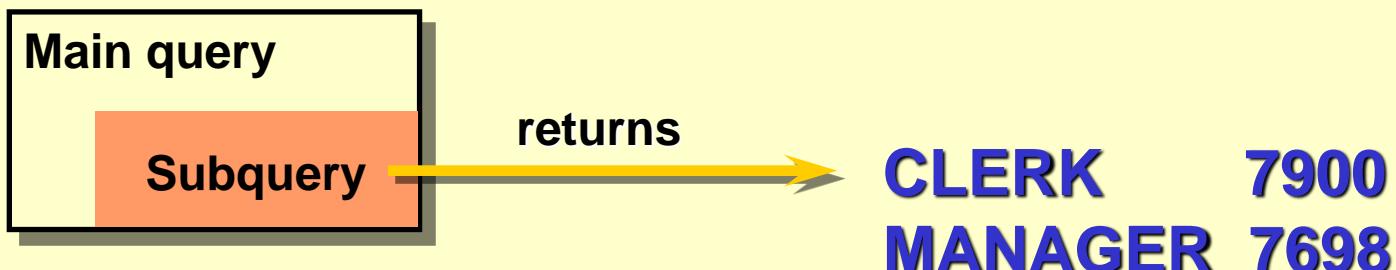
- Single-row subquery or scalar subquery



- Multiple-row subquery



- Multiple-column subquery



Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Executing Single-Row Subqueries

```
SQL> SELECT      ename, job
  2  FROM        emp
  3  WHERE       job =          ← CLERK
  4
  5
  6
  7  AND        sal >          ← 1100
  8
  9
 10
```

(SELECT job
 FROM emp
 WHERE empno = 7369)

(SELECT sal
 FROM emp
 WHERE empno = 7876);

ENAME	JOB
-----	-----
MILLER	CLERK

Using Group Functions in a Subquery

```
SQL> SELECT    ename, job, sal
  2  FROM      emp
  3 WHERE     sal =  

  4
  5
```

(SELECT MIN(sal)
FROM emp);

800

ENAME	JOB	SAL
SMITH	CLERK	800

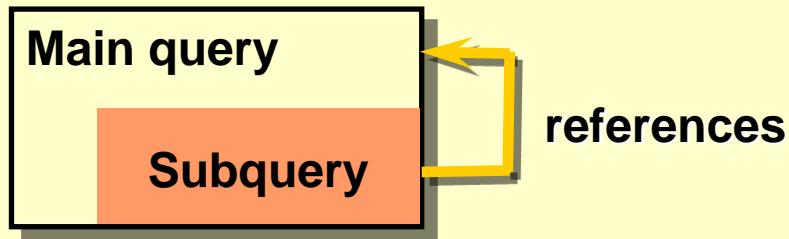
Finding second highest sal

```
SQL> SELECT MAX(sal) FROM emp  
2 WHERE sal < (SELECT MAX(sal) FROM emp);
```

```
SQL> SELECT     ename, job, sal  
2  FROM      emp  
3  WHERE      sal =  
4          (SELECT MAX(sal) FROM emp  
5  WHERE sal < (SELECT MAX(sal) FROM emp));
```

```
select * from emp order by sal desc  
OFFSET 1 ROWS FETCH NEXT 1  
ROWS ONLY;
```

Correlated Subquery



- The above variation of subquery is called **correlated subquery**, because the inner query uses a table alias of outer query.
- Unlike a simple subquery that is executed only once, a correlated subquery is execute for more than once.
- The inner query is evaluated against each row of the outer query.
- `select * from dept where not exists
(select * from emp where emp.deptno = dept.deptno)`

Finding nth highest sal

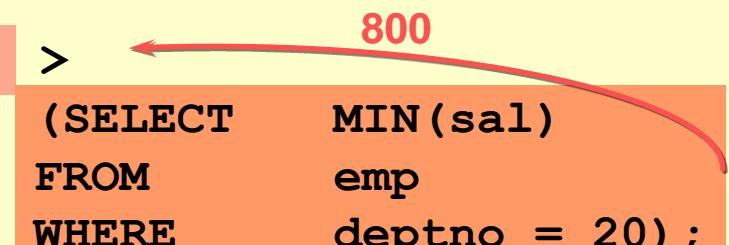
```
SELECT * FROM Emp Emp1  
WHERE (N-1) = (  
SELECT COUNT(DISTINCT Emp2.Sal)  
FROM Emp Emp2  
WHERE Emp2.Sal > Emp1.Sal);
```

- The subquery or inner query refers a column or table alias of the outer or parent query.
- Unlike a simple subquery that is executed only once, a correlated subquery is evaluated once for each row processed by the outer query.

HAVING Clause with Subqueries

- The Oracle Server executes subqueries first.
- The Oracle Server returns results into the HAVING clause of the main query.

```
SQL> SELECT      deptno, MIN(sal)
  2  FROM          emp
  3  GROUP BY      deptno
  4  HAVING        MIN(sal) > 800
  5
  6
  7
```



What Is Wrong with This Statement?

```
SQL> SELECT empno, ename  
2   FROM emp  
3 WHERE sal =  
4  
5  
6
```

```
(SELECT      MIN(sal)  
  FROM        emp  
 GROUP BY    deptno);
```

*Single-row operator with
multiple-row subquery*

ERROR:

**ORA-01427: single-row subquery returns more than
one row**

no rows selected

Will This Statement Work?

```
SQL> SELECT ename, job  
2   FROM emp  
3  WHERE job =  
4  
5  
6          (SELECT job  
      FROM emp  
     WHERE ename= 'SMYTHE' ) ;
```

```
no rows selected
```

Subquery returns no values

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery. The overall result is TRUE if it is TRUE for ANY of the values returned by the subquery.
ALL	Compare value to every value returned by the subquery. The overall result is TRUE only if it is TRUE for ALL of the values returned by the subquery.

Using IN Operator in Multiple-Row Subqueries

```
SQL> SELECT    empno, ename, job
  2  FROM      emp
  3  WHERE     sal IN      (SELECT    MIN(sal)
  4                                FROM      emp
  5                                GROUP BY deptno);
```

```
SQL> SELECT    empno, sal, deptno
  2  FROM      emp
  3  WHERE     sal < IN (800, 950, 1300);
```

EMPNO	ENAME	JOB
7369	SMITH	CLERK
7900	JAMES	CLERK
7934	MILLER	CLERK

Using ANY Operator in Multiple-Row Subqueries

```
SQL> SELECT    empno, ename, job
  2  FROM      emp
  3  WHERE     sal < ANY
  4
  5
  6
  7  AND      job <> 'CLERK';
```

```
(SELECT    sal
  FROM      emp
  WHERE     job = 'CLERK')
```

EMPNO	ENAME	JOB
7654	MARTIN	SALESMAN
7521	WARD	SALESMAN

Using ALL Operator in Multiple-Row Subqueries

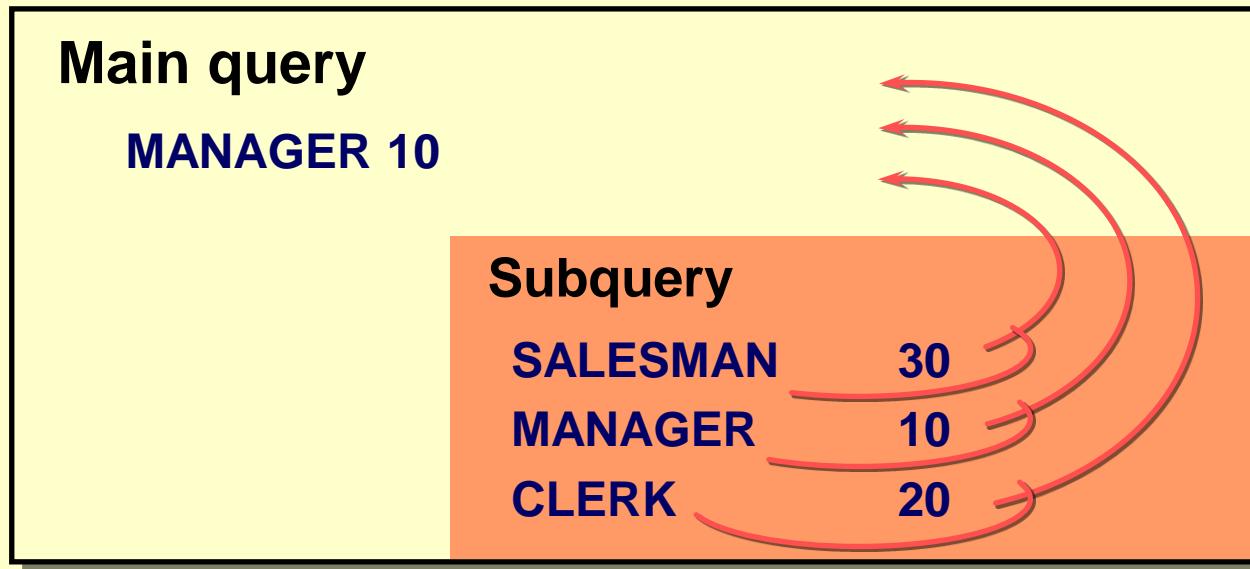
```
SQL> SELECT    emno, ename, job
  2  FROM      emp
  3 WHERE     sal > ALL
  4
  5
  6          (SELECT      avg(sal)
   FROM        emp
   GROUP BY    deptno) ;
```

EMPNO	ENAME	JOB
7839	KING	PRESIDENT
7566	JONES	MANAGER
7902	FORD	ANALYST
7788	SCOTT	ANALYST

Multiple-Column Subqueries

- After completing this lesson, you should be able to do the following:
 - Write a multiple-column subquery
 - Describe and explain the behavior of subqueries when null values are retrieved
 - Write a subquery in a FROM clause

Multiple-Column Subqueries



Main query
compares
to

Values from a multiple-row and
multiple-column subquery

MANAGER 10

SALESMAN	30
MANAGER	10
CLERK	20

Using Multiple-Column Subqueries

- Display the order number, product number, and quantity of any item in which the product number and quantity match **both** the product number and quantity of an item in order 605.

```
SQL> SELECT      ordid, prodid, qty
  2  FROM        item
  3  WHERE       (prodid, qty) IN
  4
  5
  6
  7  AND        ordid <> 605;
```

```
(SELECT prodid, qty
  FROM   item
  WHERE  ordid = 605)
```

Column Comparisons

Pairwise

PRODID	QTY
101863	100
100861	100
102130	10
100890	5
100870	500
101860	50

Nonpairwise

PRODID	QTY
101863	100
100861	100
102130	10
100890	5
100870	500
101860	50

Nonpairwise Comparison Subquery

- Display the order number, product number, and quantity of any item in which the product number and quantity match any product number and any quantity of an item in order 605.

```
SQL> SELECT      ordid, prodid, qty
  2  FROM        item
  3  WHERE       prodid IN      (SELECT      prodid
  4                                FROM
  5                                WHERE      ordid = 605)
  6  AND        qty      IN      (SELECT      qty
  7                                FROM
  8                                WHERE      ordid = 605)
  9  AND        ordid <> 605;
```

Nonpairwise Subquery

ORDID	PRODID	QTY
609	100870	5
616	100861	10
616	102130	10
621	100861	10
618	100870	10
618	100861	50
616	100870	50
617	100861	100
619	102130	100
615	100870	100
617	101860	100
621	100870	100
617	102130	100
.		

16 rows selected.

Null Values in a Subquery

```
SQL> SELECT employee.ename  
2   FROM emp employee  
3 WHERE employee.empno NOT IN  
4                               (SELECT manager.mgr  
5                                FROM emp manager);  
no rows selected.
```

Using a Subquery in the FROM Clause

```
SQL> SELECT a.ename, a.sal, a.deptno, b.salavg
  2  FROM emp a, (SELECT deptno, avg(sal) salavg
  3                      FROM emp
  4                      GROUP BY deptno) b
  5  WHERE a.deptno = b.deptno
  6  AND      a.sal > b.salavg;
```

ENAME	SAL	DEPTNO	SALAVG
KING	5000	10	2916.6667
JONES	2975	20	2175
SCOTT	3000	20	2175
...			
6 rows selected.			



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



(SQL)

Manipulating Data

Asif Sohail

**University of the Punjab
Punjab University College of Information Technology (PUCIT)**

Objectives

- After completing this lesson, you should be able to do the following:
 - Describe each DML statement
 - Insert rows into a table
 - Update rows in a table
 - Delete rows from a table
 - Control transactions

Data Manipulation Language

- A DML statement is executed when you:
 - Add new rows to a table ([INSERT statement](#))
 - Modify existing rows in a table ([UPDATE statement](#))
 - Remove existing rows from a table ([DELETE statement](#))
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

Adding a New Row to a Table

50	DEVELOPMENT	DETROIT
----	-------------	---------

New row

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

**“...insert a new row
into DEPT table...”**

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	DEVELOPMENT	DETROIT

The INSERT Statement

- 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.

```
SQL> INSERT INTO      dept (deptno, dname, loc)
  2  VALUES            (50, 'DEVELOPMENT', 'DETROIT');
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.

```
SQL> INSERT INTO      dept (deptno, dname)
  2  VALUES            (60, 'MIS');
1 row created.
```

- Explicit method: Specify the NULL keyword.

```
SQL> INSERT INTO      dept
  2  VALUES            (70, 'FINANCE', NULL);
1 row created.
```

- Inserting a row with default value(s)

Inserting Special Values

- The SYSDATE function records the current date and time.

```
SQL> INSERT INTO      emp (empno, ename, job,
          VALUES          mgr, hiredate, sal, comm, deptno)
                  (7196, 'GREEN', 'SALESMAN',
                   7782, SYSDATE, DEFAULT, NULL, 10);
1 row created.
```

Inserting Specific Date Values

- Add a new employee.

```
SQL> INSERT INTO emp
  2  VALUES          (2296, 'AROMANO', 'SALESMAN', 7782,
  3                      TO_DATE('FEB 3, 97', 'MON DD, YY'),
  4                      1300, NULL, 10);
1 row created.
```

- Verify your addition.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
2296	AROMANO	SALESMAN	7782	03-FEB-97	1300		10

Copying Rows from Another Table

- Write your INSERT statement with a subquery.

```
SQL> INSERT INTO managers(id, name, salary, hiredate)
  2          SELECT empno, ename, sal, hiredate
  3          FROM   emp
  4          WHERE  job = 'MANAGER';
  5
  6 3 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

EMP

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		10
7566	JONES	MANAGER		20
...				

“...update a row
in EMP table...”



EMP

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		20
7566	JONES	MANAGER		20
...				

The UPDATE Statement

- 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 when you specify the WHERE clause.

```
SQL> UPDATE    emp  
  2 SET        deptno = 20  
  3 WHERE      empno = 7782;  
1 row updated.
```

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

```
SQL> UPDATE    employee  
  2 SET        deptno = 20;  
14 rows updated.
```

Updating with Multiple-Column Subquery

- Update employee 7698's job and department to match that of employee 7499.

```
SQL> UPDATE emp
  2 SET      (job, deptno) =
  3                               (SELECT job, deptno
  4                                FROM   emp
  5                               WHERE  empno = 7499)
  6 WHERE    empno = 7698;
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.

```
SQL> UPDATE employee
  2  SET deptno = (SELECT deptno
  3                      FROM emp
  4                     WHERE empno = 7369)
  5  WHERE job      = (SELECT job
  6                      FROM emp
  7                     WHERE empno = 7369);
2 rows updated.
```

Updating Rows: Integrity Constraint Error

```
SQL> UPDATE emp  
  2  SET deptno = 55  
  3  WHERE deptno = 10;
```

```
UPDATE emp  
      *  
ERROR at line 1: •Department number 55 does not exist  
ORA-02291: integrity constraint (USR.EMP_DEPTNO_FK)  
violated - parent key not found
```

Inserting or Updating Rows Based on DEFAULT value

- 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;
```

Removing a Row from a Table

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	DEVELOPMENT	DETROIT
60	MIS	
...		

**“...delete a row
from DEPT table...”**

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
60	MIS	
...		

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 when you specify the WHERE clause.

```
SQL> DELETE FROM      department  
  2  WHERE              dname = 'DEVELOPMENT';  
1 row deleted.
```

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

```
SQL> DELETE FROM      department;  
4 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.

```
SQL> DELETE FROM employee
  2 WHERE deptno =
  3
  4
  5
  6 rows deleted.
```

```
(SELECT deptno
   FROM dept
  WHERE dname = 'SALES' );
```

Deleting Rows: Integrity Constraint Error

```
SQL> DELETE FROM dept  
  2 WHERE deptno = 10;
```

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

```
DELETE FROM dept  
      *  
ERROR at line 1:  
ORA-02292: integrity constraint (USR.EMP_DEPTNO_FK)  
violated - child record found
```

Truncating a Table

- The TRUNCATE TABLE statement:
 - Removes all rows from a table
 - Releases the storage space used by that table
 - You can do the same with DELETE statement, which is a DML statement and hence recoverable, but it is slow.

```
SQL> TRUNCATE TABLE department;  
Table truncated.
```

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

COMMIT, ROLLBACK and SAVEPOINT

- Statements for transaction management.
- A transaction begins with the first DML statement or by using an explicit **SET TRANSACTION** statement and ends when a **COMMIT** or **ROLLBACK** statement is issued.
- Use COMMIT and ROLLBACK SQL statements to terminate a transaction explicitly.
- The syntax for COMMIT statement is:
 - **COMMIT [WORK][COMMENT text];**
- The optional word WORK is used just to improve readability.
- The COMMENT is usually employed by distributed transactions.
 - **COMMIT COMMENT 'Employees details have been changed';**

COMMIT, ROLLBACK and SAVEPOINT

- SVAEPOINT gives more control on transaction management.
- SVAEPOINT marks and saves the current point in a transaction.
- This allows you to rollback the changes up to that point.
- There can be maximum 5 savepoints in a transaction.
- The syntax for SAVEPOINT statement is:
 - SVAEPOINT savepoint-name;
 - SVAEPOINT Emp_savepoint;

COMMIT, ROLLBACK and SAVEPOINT

- ROLLBACK statement is used to undo the changes made by the transaction. However, it can't undo the changes that have been made permanent by COMMIT statement.
- The syntax for ROLLBACK statement is:
 - `ROLLBACK [WORK][TO savepoint_name];`
- Examples:
- `ROLLBACK;` // it will undo the changes made by the current transaction (up to the last COMMIT)
- `ROLLBACK TO rollback_point;`
- // it will undo the changes up to the specific savepoint.

COMMIT, ROLLBACK and SAVEPOINT (Example)

Transaction	Time
t1	SAVEPOINT s1; ----- (a) ----- (b)
t2	SAVEPOINT s2; ----- (c) ----- (d)
t3	SAVEPOINT s3; ----- (e) ----- (f)
t4	ROLLBACK TO s2; ----- (g) ----- (h)
t5	COMMIT; // work done since the start of the transaction plus (g) and (h) will be saved.



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



(SQL)

Creating and Managing Tables

Asif Sohail

University of the Punjab
Punjab University College of Information Technology (PUCIT)

Note:

The slides are adapted from Oracle corporation's slides.

Objectives

- After completing this lesson, you should be able to do the following:
 - Describe the main database objects
 - Create tables
 - Describe the data types that can be used when specifying column definition
 - Alter table definitions
 - Drop, rename, and truncate tables

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	Generates primary key values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

Naming Conventions

- Must begin with a letter
- Can be 1–30 characters long
- Can contain only A–Z, a–z, 0–9, _, \$, and #
- Names are not case sensitive (enclose names in double quotes to make them case sensitive)
- Must not duplicate the name of another object owned by the same user
- Must not be an Oracle Server reserved word

The CREATE TABLE Statement

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

```
CREATE TABLE table_name  
(col_name1 datatype(size) [Default expr] [Col Constraints],  
 col_name2 datatype(size) [Default expr] [Col Constraints],  
 .....  
 [Table Constraints]);
```

You specify:

- Table name
- Column name, column data type, and column size
- Size specifies the max. length of a column.
- An oracle table can have up to 256 columns.

Data Types

Datatype	Description
VARCHAR2(size)	Variable-length character data
CHAR(size)	Fixed-length character data
NUMBER(p,s)	Variable-length numeric data
DATE	Date and time values

Creating Tables

```
SQL> CREATE TABLE dept  
  2       (deptno NUMBER(2),  
  3         dname  VARCHAR2(14),  
  4         loc    VARCHAR2(13));
```

Table created.

- Confirm table creation.

```
SQL> DESCRIBE dept
```

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

Tables in the Oracle Database

- User Tables
 - Collection of tables created and maintained by the user
 - Contain user information
- Data Dictionary
 - Collection of tables created and maintained by the Oracle server
 - Contain database information

Querying the Data Dictionary

- Describe tables owned by the user.

```
SQL> SELECT *  
2  FROM    user_tables;
```

- View distinct object types owned by the user.

```
SQL> SELECT   DISTINCT object_type  
2  FROM    user_objects;
```

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

```
SQL> SELECT *  
2  FROM    user_catalog;
```

Including Constraints

- Constraints enforce data integrity rules.
- Constraint can be either at table level or column level.
- A constraint should be given a meaningful name (its optional though), otherwise oracle server give every constraint a unique name in sys-cn format.
- A constraint can be defined either at the time of table creation or after it.
- The constraints are stored in a data dictionary view called **USER_CONSTRAINTS**.
- The following constraint types are valid in Oracle:
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK

Including Constraints

- The following constraint types are valid in Oracle:
 - NOT NULL [Column Level]
 - UNIQUE [Column or Table Level]
 - PRIMARY KEY [Column or Table Level]
 - FOREIGN KEY [Column or Table Level]
 - CHECK [Column or Table Level]
 - DEFAULT [Column Level]

Defining Constraints

```
CREATE TABLE tablename
(col_name1 datatype(size) [Default expr] [Col Constraints],
.....
[table_constraint] [, . . . ]) ;
```

```
CREATE TABLE emp (
    empno  NUMBER(4) ,
    ename   VARCHAR2(10) ,
    ...
    deptno NUMBER(7,2) NOT NULL,
    CONSTRAINT emp_empno_pk
                PRIMARY KEY (EMPNO) );
```

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

EMP

EMPNO	ENAME	JOB	...	COMM	DEPTNO
7839	KING	PRESIDENT			10
7698	BLAKE	MANAGER			30
7782	CLARK	MANAGER			10
7566	JONES	MANAGER			20
...					

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

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

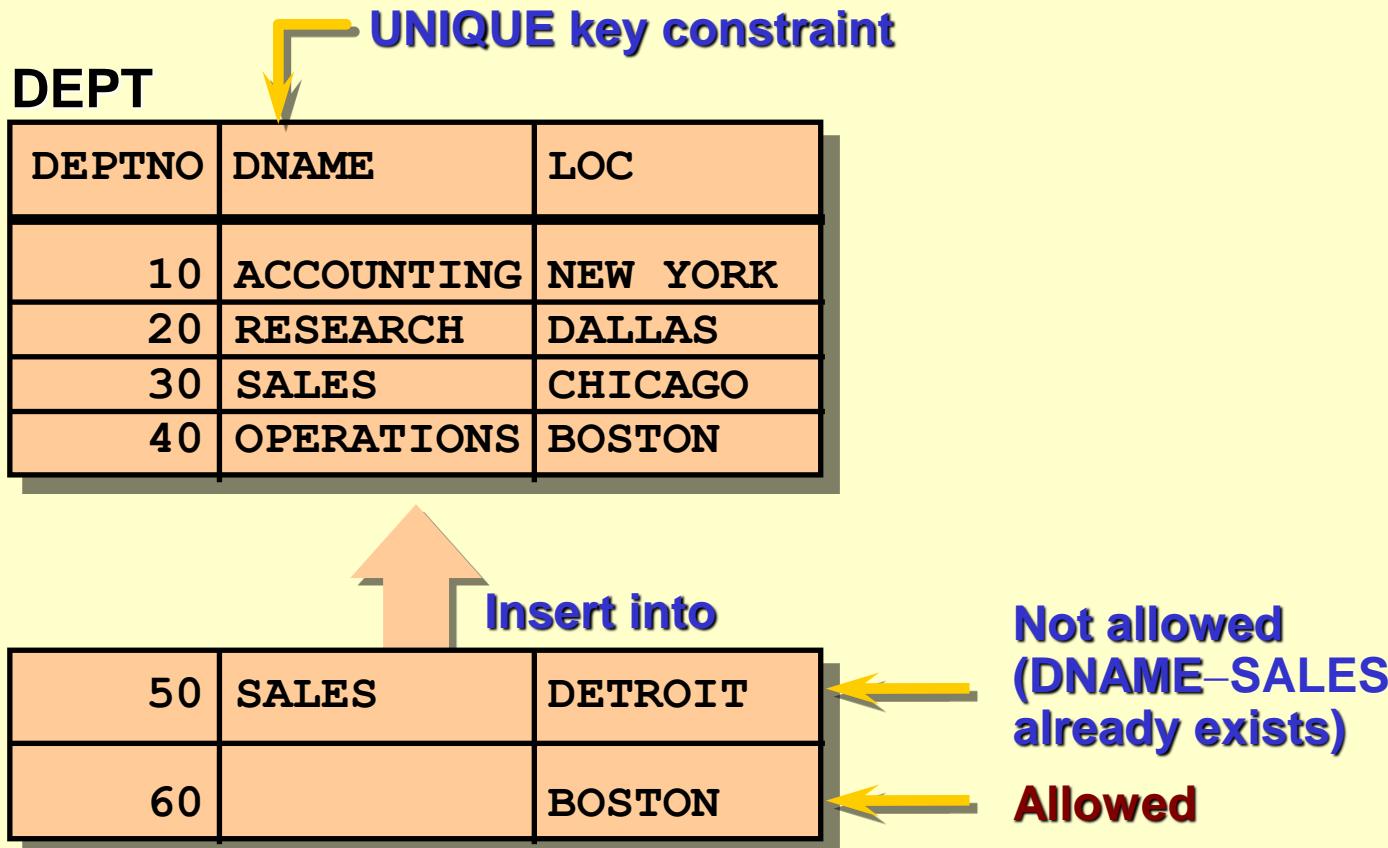
NOT NULL constraint

The NOT NULL Constraint

- Defined at the column level

```
SQL> CREATE TABLE emp (
  2      empno      NUMBER(4) ,
  3      ename      VARCHAR2(10) NOT NULL,
  4      job        VARCHAR2(9) ,
  5      mgr        NUMBER(4) ,
  6      hiredate   DATE ,
  7      sal         NUMBER(7,2) ,
  8      comm       NUMBER(7,2) ,
  9      deptno     NUMBER(7,2) NOT NULL) ;
```

The UNIQUE Key Constraint

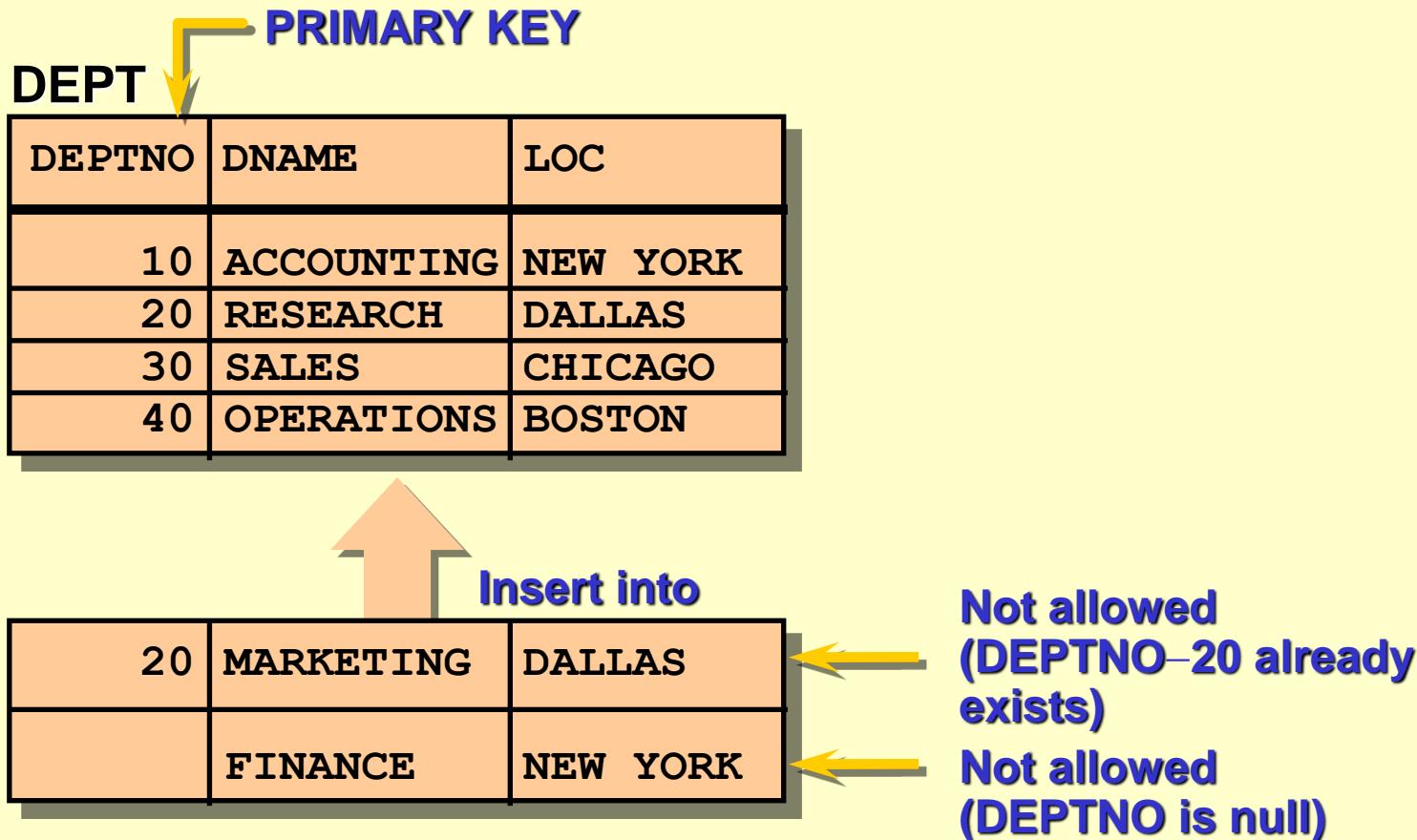


The UNIQUE Key Constraint

- Defined at either the table level or the column level

```
SQL> CREATE TABLE dept(
  2      deptno      NUMBER(2) ,
  3      dname        VARCHAR2(14) ,
  4      loc          VARCHAR2(13) ,
  5      CONSTRAINT dept_dname_uk UNIQUE(dname) );
```

The PRIMARY KEY Constraint

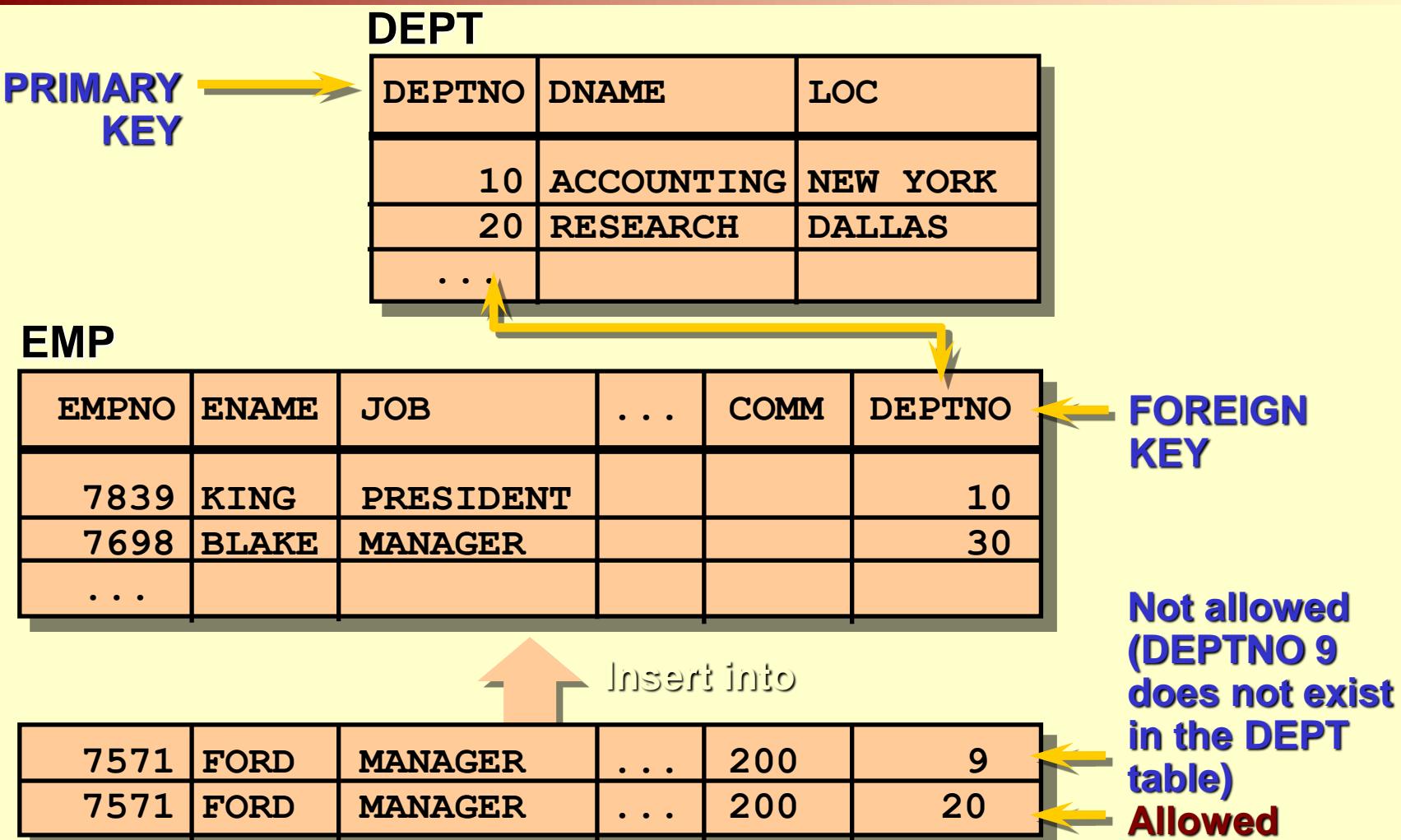


The PRIMARY KEY Constraint

- Defined at either the table level or the column level, however if the PK is composite, then it should be defined at table level.

```
SQL> CREATE TABLE dept(
 2      deptno      NUMBER(2) ,
 3      dname       VARCHAR2(14) ,
 4      loc         VARCHAR2(13) ,
 5      CONSTRAINT dept_dname_uk UNIQUE (dname) ,
 6      CONSTRAINT dept_deptno_pk PRIMARY KEY(deptno)) ;
```

The FOREIGN KEY Constraint



The FOREIGN KEY Constraint

- Defined at either the table level or the column level

```
SQL> CREATE TABLE emp (
  2      empno      NUMBER(4),
  3      ename      VARCHAR2(10) NOT NULL,
  4      job        VARCHAR2(9),
  5      mgr        NUMBER(4),
  6      hiredate   DATE,
  7      sal        NUMBER(7,2),
  8      comm       NUMBER(7,2),
  9      deptno    NUMBER(7,2) NOT NULL,
 10      CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno)
 11          REFERENCES dept (deptno);
```

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

it is optional clause and it Allows deletion in the parent table and deletion of the dependent rows in the child table

- ON DELETE SET NULL:

Converts dependent foreign key values to null

The CHECK Constraint

- Defines a condition that each row must satisfy

```
..., deptno NUMBER(2),  
CONSTRAINT emp_deptno_ck  
CHECK (DEPTNO BETWEEN 10 AND 99),...
```

```
..., program CHAR(5),  
CONSTRAINT student_program_ck  
CHECK (program IN ('BS', 'MIT', 'MS')),...
```

The DEFAULT Option

- To specify a default value for a column.
- The default data type must match the column data type.

```
... hire_date DATE DEFAULT SYSDATE, ...
... Status CHAR(1) DEFAULT 'P', ...
```

```
SQL> CREATE TABLE test
  2       (id      NUMBER(2) ,
  3        value   NUMBER(2) DEFAULT 50,
  4        name    VARCHAR2(13)) ;
```

Table created.

Viewing Constraints

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

```
SQL> SELECT constraint_name, constraint_type,  
2          search_condition  
3      FROM user_constraints  
4     WHERE table_name = 'EMP';
```

CONSTRAINT_NAME	C SEARCH_CONDITION
SYS_C00674	C EMPNO IS NOT NULL
SYS_C00675	C DEPTNO IS NOT NULL
EMP_EMPNO_PK	P
...	

Viewing the Columns Associated with Constraints

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

```
SQL> SELECT constraint_name, column_name  
2   FROM user_cons_columns  
3  WHERE table_name = 'EMP';
```

CONSTRAINT_NAME	COLUMN_NAME
EMP_DEPTNO_FK	DEPTNO
EMP_EMPNO_PK	EMPNO
EMP_MGR_FK	MGR
SYS_C00674	EMPNO
SYS_C00675	DEPTNO

Creating a Table by Using a Subquery

- Create a table and insert rows by combining the CREATE TABLE statement and 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.
- To create new table without populating it with the data in the existing table, use the following:

```
CREATE TABLE table
    [(column, column...)]
AS subquery WHERE 1=2;
```

Creating a Table by Using a Subquery

```
SQL> CREATE TABLE    dept30
  2  AS
  3      SELECT      empno, ename, sal*12 ANNSAL, hiredate
  4      FROM        emp
  5      WHERE       deptno = 30;
```

Table created.

```
SQL> DESCRIBE dept30
```

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
ANNSAL		NUMBER
HIREDATE		DATE

Creating a Small Database (Exercise)

Table Name: Student	
Col Name	Constraints
RegNo	Primary Key
SName	Not Null
Program	Can be 'BS', 'BBA', or 'BA'
NIC	Unique

Table Name: Course	
Col Name	Constraints
CourseCode	Primary Key
CourseTitle	Not Null
CreditHours	Can be either 3 or 4

Table Name: Result	
Col Name	Constraints
RegNo	Primary Key, Foreign Key
CourseCode	Primary Key, Foreign Key
CourseMarks	Between 0 and 100

The ALTER TABLE Statement

- Use the ALTER TABLE statement to:
 - Add a new column
 - Modify an existing column
 - Drop a column
 - Add or Drop a Constraint.
 - Disable or Enable a Constraint

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

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

Adding a Column

DEPT30

EMPNO	ENAME	ANNSAL	HIREDATE	JOB
7698	BLAKE	34200	01-MAY-81	
7654	MARTIN	15000	28-SEP-81	
7499	ALLEN	19200	20-FEB-81	
7844	TURNER	18000	08-SEP-81	
...				

New column

“...add a new column into DEPT30 table...”

DEPT30

EMPNO	ENAME	ANNSAL	HIREDATE	JOB
7698	BLAKE	34200	01-MAY-81	
7654	MARTIN	15000	28-SEP-81	
7499	ALLEN	19200	20-FEB-81	
7844	TURNER	18000	08-SEP-81	
...				

Adding a Column

- You use the ADD clause to add columns.

```
SQL> ALTER TABLE dept30
  2 ADD          (job VARCHAR2(9));
Table altered.
```

- The new column becomes the last column.

EMPNO	ENAME	ANNSAL	HIREDATE	JOB
7698	BLAKE	34200	01-MAY-81	
7654	MARTIN	15000	28-SEP-81	
7499	ALLEN	19200	20-FEB-81	
7844	TURNER	18000	08-SEP-81	
...				
6 rows selected.				

Modifying a Column

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

```
ALTER TABLE    dept30
MODIFY        (ename VARCHAR2(15));
Table altered.
```

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

- ALTER TABLE t READ ONLY

Dropping and Renaming a Column

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

```
ALTER TABLE dept30
DROP COLUMN Job;
Table altered.
```

- Use the RENAME COLUMN clause to rename a col.

```
ALTER TABLE dept30
RENAME COLUMN Job TO Designation;
Table altered.
```

Adding a Constraint

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

- Add a FOREIGN KEY constraint to the EMP table indicating that a manager must already exist as a valid employee in the EMP table.

```
SQL> ALTER TABLE      emp
  2 ADD CONSTRAINT   emp_mgr_fk
  3          FOREIGN KEY(mgr) REFERENCES emp(empno);
Table altered.
```

Dropping a Constraint

- Remove foreign key constraint from the EMP table.

```
SQL> ALTER TABLE      emp
  2  DROP CONSTRAINT  emp_mgr_fk;
Table altered.
```

- Remove the PRIMARY KEY constraint on the DEPT table and drop the associated FOREIGN KEY constraint on the EMP.DEPTNO column.

```
SQL> ALTER TABLE      dept
  2  DROP PRIMARY KEY CASCADE;
Table altered.
```

- To drop a DEFAULT constraint

```
SQL> ALTER TABLE      student
  2  ALTER COLUMN program DROP DEFAULT;
Table altered.
```

Disabling Constraints

- Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint.
- Apply the CASCADE option to disable dependent integrity constraints.

```
SQL> ALTER TABLE emp  
  2  DISABLE CONSTRAINT emp_empno_pk CASCADE;  
Table altered.
```

Enabling Constraints

- Activate an integrity constraint currently disabled in the table definition by using the ENABLE clause.

```
SQL> ALTER TABLE emp  
  2  ENABLE CONSTRAINT emp_empno_pk;  
Table altered.
```

- A UNIQUE or PRIMARY KEY index is automatically created if you enable a UNIQUE key or PRIMARY KEY constraint.

Changing the Name of an Object

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

```
SQL> RENAME dept TO department;  
Table renamed.
```

- CREATE GLOBAL TEMPORARY TABLE

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 this statement.

```
SQL> DROP TABLE dept30;  
Table dropped.
```

- NOTE: if the table has a reference in another table, the above command may not work. To drop all the references as well, use the following:

```
SQL> DROP TABLE dept30 CASCADE CONSTRAINTS;  
Table dropped.
```



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



Introduction to PL / SQL

Asif Sohail

**University of the Punjab
Punjab University College of Information Technology (PUCIT)**

Introduction

- PL/SQL stands for Procedural Language/SQL.
- PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL.
- The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.
- A block has the following structure:

Structure of a Block

DECLARE

```
/* Declarative section - Optional */
```

BEGIN

```
/* Executable section: procedural and SQL statements  
go here. This is the only section of the block that  
is required. */
```

EXCEPTION

```
/* Exception handling section: error handling  
statements go here. (Optional) */
```

END;

Coding a Block

- Only the executable section is required. The other sections are optional.
- The only SQL statements allowed in a PL/SQL program DML statements such as SELECT, INSERT, UPDATE, DELETE plus some transaction control.
- Data definition statements like CREATE, DROP, or ALTER are not allowed.
- The executable section also contains constructs such as assignments, branches, loops, procedure calls, and triggers.
- PL/SQL is not case sensitive.
- C style comments are used.
 - This is a single line comment
 - /* This is a multiline comment */

Variables and Types

- Information is transmitted between a PL/SQL program and the database through variables.
- Every variable has a specific type associated with it.
- Types in PL/SQL can be tricky. In many cases, a PL/SQL variable will be used to manipulate data stored in an existing relation. In this case, it is essential that the variable have the same type as the relation column. If there is any type mismatch, variable assignments and comparisons may not work the way you expect.
- To be safe, instead of hard coding the type of a variable, you should use the %TYPE operator. For example:
- `id emp.empno%TYPE;`
- gives PL/SQL variable id whatever type was declared for the empno column in relation emp.

Initializing a Variable

- The initial value of any variable, regardless of its type, is NULL.
- We can assign values to variables, using the ":=" operator or using **DEFAULT** keyword.
- The assignment can occur either immediately after the type of the variable is declared, or anywhere in the executable portion of the program.
- **Example**

DECLARE

X NUMBER := 10;

Greetings varchar2(10) DEFAULT 'Welcome';

Sample Programs

- Type SET SERVEROUTPUT ON at SQL prompt before writing the program, so that the output of the program can be seen at the display.
- **Program 1 /*To Display a message on the screen*/**

```
BEGIN  
DBMS_OUTPUT.PUT_LINE ('Hello World');  
END;  
/
```

Sample Programs

- Program 2 /*To Display the value of a variable*/

DECLARE

a NUMBER := 3;

b CONSTANT NUMBER :=100;

BEGIN

a := a + 10;

DBMS_OUTPUT.PUT_LINE (a);

DBMS_OUTPUT.PUT_LINE (b);

END;

/

Sample Programs

- Suppose we have table named **temp** with 2 attribute **x** number and **y** char.
- **Program 3 /*To insert a record in the table*/**

BEGIN

INSERT INTO TEMP VALUES(1,'A');

END;

/

Sample Programs

- Suppose we have table named **temp** with 2 attribute **x number** and **y char**.
- **Program 4 /*To insert a record in the table using variables*/**

```
DECLARE  
    NUM NUMBER := 2;  
    CH CHAR := 'B';  
BEGIN  
    INSERT INTO TEMP VALUES(NUM,CH);  
END;  
/
```

Sample Programs

- Program 5 /*To input a values and display it*/

```
DECLARE
    NUM NUMBER;;
    CH CHAR;
BEGIN
    NUM := &NUM;
    CH := '&CH';
    DBMS_OUTPUT.PUT_LINE ('NUM = ' || NUM);
    DBMS_OUTPUT.PUT_LINE ('CH = ' || CH);
END;
/
```

SELECT statement is PL/SQL

- The SELECT statement has a special form in which a single tuple is placed in variables.
- After the SELECT clause, we must have an INTO clause listing variables, one for each attribute in the SELECT clause, into which the components of the retrieved tuple must be placed.
- Example:
- `select empno, ename into id, name from emp where empno=7536;`
- id and name are the variables declared in the declaration section.

PL/SQL Operators

- Arithmetic operators: +, -, *, /, **(exponent)
- Relational operators: =, != <>, <, >, <=, >=
- Logical operators: AND, OR, NOT
- Comparison operators: LIKE, BETWEEN, IN, IS NULL

PL/SQL Conditions

<pre>IF (condition) THEN S; END IF;</pre>	<pre>IF condition THEN S1; ELSE S2; END IF;</pre>
<pre>IF(boolean_expression 1)THEN S1; ELSIF(boolean_expression 2) THEN S2; ELSIF(boolean_expression 3) THEN S3; ELSE S4; END IF;</pre>	<pre>CASE selector WHEN 'value1' THEN S1; WHEN 'value2' THEN S2; WHEN 'value3' THEN S3; ... ELSE Sn; -- default case END CASE;</pre>

PL/SQL Loops

LOOP

Statement(s);

EXIT / EXIT WHEN statement

END LOOP;

WHILE condition LOOP

Statement(s);

END LOOP;

FOR counter IN [REVERSE] initial_value .. final_value LOOP

sequence_of_statements;

END LOOP;

Continue, Exit, Exit When statements

Procedures

- PL/SQL procedures behave very much like procedures in other programming language.
- **Declaring a Procedure**

```
CREATE [OR REPLACE] PROCEDURE procedure-name  
[(argument1 ... [, argumentN) ]  
{IS | AS}  
[local-variable-declarations]  
BEGIN  
executable-section  
[exception-section]  
END [procedure-name];
```

Procedures

- Unlike the type specifier in a PL/SQL variable declaration, the type specifier in a parameter declaration must be unconstrained.
- For example, CHAR(10) and VARCHAR(20) are illegal; CHAR or VARCHAR should be used instead.
- The actual length of a parameter depends on the corresponding argument that is passed in when the procedure is invoked.

Procedures

- **Example: A procedure to add two numbers.**

create or replace procedure add is

a NUMBER := 3;

b NUMBER := 6;

begin

DBMS_OUTPUT.PUT_LINE (a + b);

end;

- To execute the above procedure; type
execute add at SQL prompt.

Procedures

- **Example: A procedure to add two numbers using parameters.**

```
create or replace procedure add(a number, b number) is  
begin
```

```
    DBMS_OUTPUT.PUT_LINE (a + b);
```

```
end;
```

- To execute the above procedure; type
execute add(5,10) at SQL prompt.

Procedures

- **Example: A procedure to insert a record in temp table.**

create or replace procedure insert_record (a number,
b char) is

```
begin  
insert into temp values(a , b);  
end;
```

- To execute the above procedure; type
`execute insert_record(1 , 'A')` at SQL prompt.

Procedures

- **Example: A procedure to insert a record in temp table.**

```
create or replace procedure insert_record (a temp.x%type,  
b temp.y%type) is
```

```
begin  
insert into temp values(a , b);  
end;
```

- To execute the above procedure; type
`execute insert_record(2 , 'B')` at SQL prompt.

Functions

- A PL/SQL function declaration is similar to a procedure declaration--;except that the function returns a value of a predefined data type.
- **Declaring a Function**

```
CREATE [OR REPLACE] FUNCTION function-name  
[(argument1 ... [, argumentN) ]  
RETURN function-datatype IS  
[local-variable-declarations]  
BEGIN  
executable-section  
[exception-section]  
END [function-name];
```

Functions - Example

- Consider a table student(rollno, sname, marks).
- The purpose of the following function is to determine whether the student is PASS or FAIL.

```
create or replace function compute_grade(a number)
return char is
m student.marks%type;
begin
select marks into m from result where rollno=a;
if m>=50 then return 'Pass';
else return 'Fail';
end if;
end;
```

Calling a Functions

- Since the function returns a value, therefore it must either be called in a select statement OR in an expression.
- To call the above function, type at SQL prompt;
- Select compute_grade(1) from dual; // 1 is the rollno of the student.

Obtaining Error Messages When Creating Stored Procedures & Functions

- If Oracle detects errors when you create a stored PL/SQL program, it issues a nondescriptive message indicating that errors occurred--;without providing any additional details.
- To view the errors resulting from the attempted compilation of the PL/SQL code, you can use the SQL*Plus command `show errors`, which displays the specific PL/SQL compilation errors.
- When PL/SQL compiles a subprogram, the resulting compilation errors are stored in an Oracle data dictionary view `USER_ERRORS`.

Database Triggers

- A database trigger is a block of PL/SQL statements that are executed when a DML statement (DELETE, UPDATE, or INSERT statement) is applied to a table.
- Database triggers are used for the following purposes:
 - Change a column value based on the value of other columns in the same table or a different table
 - Perform complex validation of column values; for instance, you might need to compare a column value to some aggregate column value from a different table
 - Transaction logging
- Unlike a stored procedure, database triggers are fired or executed implicitly.

Database Triggers

- The syntax of creating a trigger is:

```
CREATE [OR REPLACE] TRIGGER trigger-name {BEFORE | AFTER}  
triggering-event ON table-name  
[FOR EACH ROW]  
[WHEN (condition)]  
PL/SQL block
```

- triggering-event is either INSERT, UPDATE, or DELETE
- FOR EACH ROW, when used, causes the trigger to fire for each affected row and the trigger is called **row level trigger**. When FOR EACH ROW is omitted, the trigger is called **statement level trigger**.
- **Twelve** possible trigger types!

Database Triggers

- Within the trigger body, a row-level trigger can reference the column values of the row that existed when the trigger was fired.
- For an **INSERT** statement, the values that will be inserted are contained in **:new.column-name** where column-name is a column in the table.
- For an **UPDATE** statement, the original value for a column is contained in **:old.column-name**; the new value for a column is contained in **:new.column-name**.
- For a **DELETE** statement, the column values of the row being deleted are contained in **:old.column-name**.

Database Triggers – Example 1

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE UPDATE ON empsal
FOR EACH ROW
DECLARE
    sal_diff number;
BEGIN
    sal_diff := :NEW.sal - :OLD.sal;
    dbms_output.put_line('Old salary: ' || :OLD.sal);
    dbms_output.put_line('New salary: ' || :NEW.sal);
    dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

Database Triggers

Acc_master

Fields	Date Type	Constraints
AccNo	Number(4)	Primary Key
AccBal	Number(10,2)	

Acc_trans

Fields	Date Type	Constraints
AccNo	Number(4)	PK, FK
transdate	Date	Primary Key
transtype	Char(1)	'D' or 'W'
Amount	Number(10,2)	

```
create or replace trigger acc_trigger
before insert on acc_trans
for each row
declare
    bal acc_master.accbal%type;
begin
    select accbal into bal from acc_master where accno = :new.accno;
    if (:new.amount>bal) then
        begin
            dbms_output.put_line('Insufficient Balance');
            :new.comments := 'Insufficient Balance';
        end;
    else
        begin
            update acc_master
            set accbal = accbal - :new.amount;
            :new.comments := 'Withdraw Successful';
        end;
    end if;
else
    begin
        update acc_master
        set accbal = accbal + :new.amount;
        :new.comments := 'Deposit Successful';
    end;
end if;
end;
```

Database Triggers (Conditional)

```
create or replace trigger Shipment_Ins_Before  
before insert on Shipment  
for each row  
when (to_number(to_char(sysdate,'HH24')) > 17)  
begin
```

```
    :new.Manual_Check := 'Y';
```

```
end;
```

- When a row is inserted into the Shipment table after 5 PM, the **Manual_Check** column is set to Y.
- The **Manual_Check** column indicates whether a shipping clerk should verify by phone the accuracy of the shipping request.

Restrictions on Triggers

- You cannot execute a COMMIT or ROLLBACK statement in a database trigger.
- A trigger may not call a stored procedure, function, or package subprogram that performs a COMMIT or ROLLBACK.

Reason

- If a trigger encounters an error, all database changes that have been propagated by the trigger should be rolled back. But if the trigger committed some portion of those database changes, Oracle would not be able to roll back the entire transaction.

Dropping, Enabling, and Disabling Triggers

- `DROP TRIGGER trigger-name;`
- `ALTER TRIGGER trigger-name DISABLE;`
- `ALTER TRIGGER trigger-name ENABLE;`
- `SELECT * FROM USER_TRIGGERS;`



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



(SQL)

Creating Views

Asif Sohail

**University of the Punjab
Punjab University College of Information Technology (PUCIT)**

Objectives

- After completing this lesson, you should be able to do the following:
 - Describe a view
 - Create a view
 - Retrieve data through a view
 - Alter the definition of a view
 - Insert, update, and delete data through a view
 - Drop a view

What Is a View?

- A View is a subset of the database that is presented to one or more users.
- A view is often referred to as a **virtual table**.
- Views are created by using SELECT statement that retrieves data from existing table(s).

What Is a View?

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
DEPTNO						
-						
7839	KING	PRESIDENT		17-NOV-81	5000	
10						
7782	CLARK	MANAGER	7839	09-JUN-81	1500	300
20						
EMPNO	ENAME	JOB				
7839	KING	PRESIDENT				
7782	CLARK	MANAGER				
7934	MILLER	CLERK				
20						
7876	ADAMS	CLERK	7788	12-JAN-83	1100	
20						
7369	SMITH	CLERK	7902	17-DEC-80	800	
20						

EMPVU10 View

EMPNO	ENAME	JOB
7839	KING	PRESIDENT
7782	CLARK	MANAGER
7934	MILLER	CLERK

Why Use Views?

- To restrict database access
- Converting between units.
- Columns can be renamed.
- Simplifying the construction of complex queries.
- To present different views of the same data
- Providing user security functions.

Creating a View

- You embed a subquery within the CREATE VIEW statement.

```
CREATE [OR REPLACE] VIEW view
  [(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION]
[WITH READ ONLY]
```

- The subquery can contain complex SELECT syntax.
- The subquery cannot contain an ORDER BY clause.

Creating a View

- Create a view, EMPVU10, that contains details of employees in department 10.

```
SQL> CREATE VIEW      empvu10
  2  AS SELECT          empno, ename, job
  3  FROM               emp
  4  WHERE              deptno = 10;
View created.
```

- Describe the structure of the view by using the SQL*Plus DESCRIBE command.

```
SQL> DESCRIBE empvu10
```

Creating a View

- Create a view by using column aliases in the subquery.

```
SQL> CREATE VIEW      salvu30
  2 AS SELECT          empno EMPLOYEE_NUMBER, ename NAME,
  3                  sal SALARY
  4 FROM             emp
  5 WHERE            deptno = 30;
View created.
```

- Select the columns from this view by the given alias names.

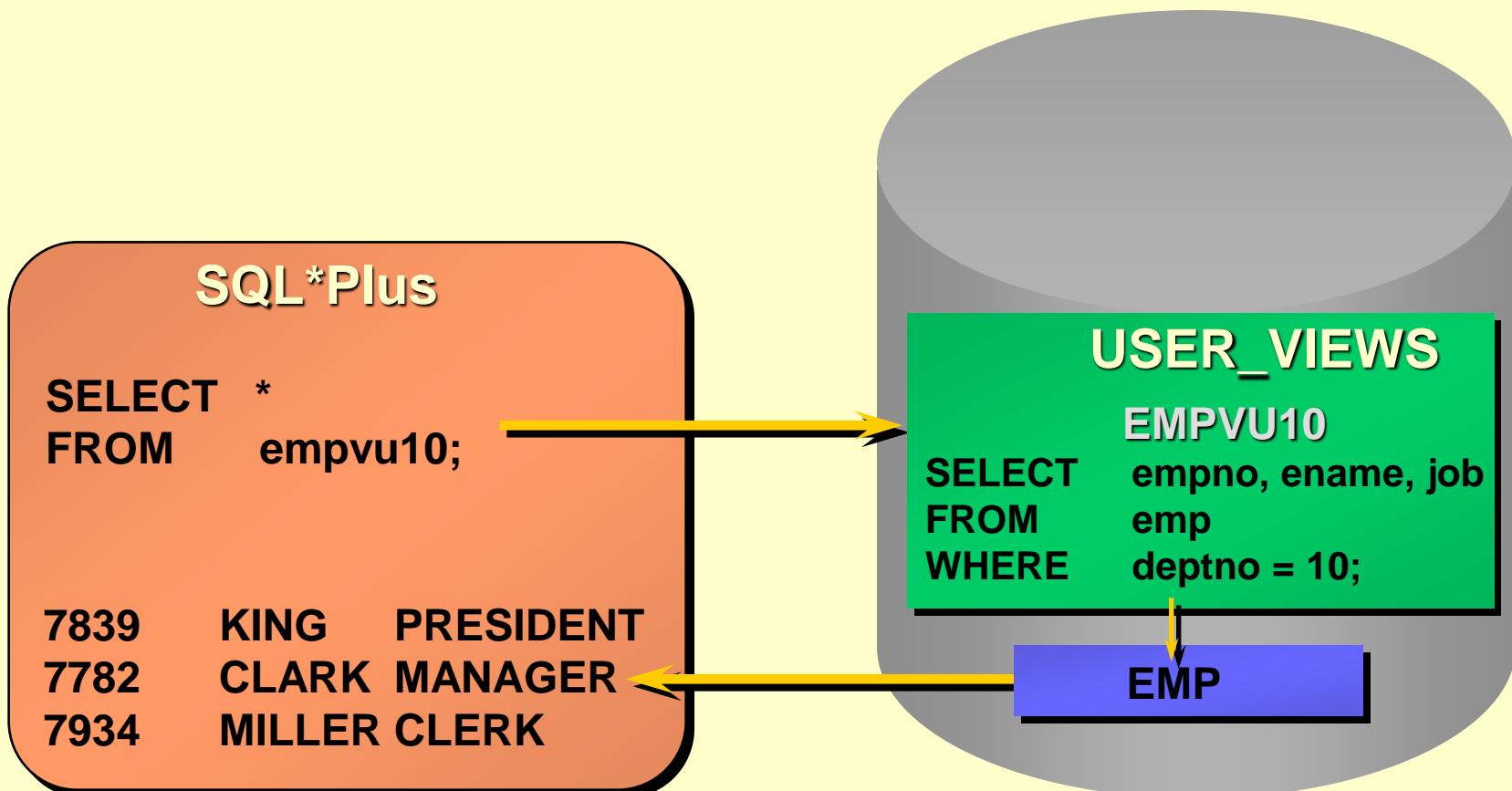
Retrieving Data from a View

```
SQL> SELECT *
2   FROM    salvu30;
```

<u>EMPLOYEE_NUMBER</u>	NAME	SALARY
7698	BLAKE	2850
7654	MARTIN	1250
7499	ALLEN	1600
7844	TURNER	1500
7900	JAMES	950
7521	WARD	1250

6 rows selected.

Querying a View



Modifying a View

- Modify the EMPVU10 view by using CREATE OR REPLACE VIEW clause. Add an alias for each column name.

```
SQL> CREATE OR REPLACE VIEW empvu10
  2      (employee_number, employee_name, job_title)
  3  AS SELECT      empno, ename, job
  4  FROM          emp
  5  WHERE         deptno = 10;
```

View created.

- Column aliases in the CREATE VIEW clause are listed in the same order as the columns in the subquery.

Simple Views and Complex Views

•Feature	Simple Views	Complex Views
•Number of tables	One	One or more
•Contain functions	No	Yes
•Contain groups of data	No	Yes
•DML through view	Yes	Not always

Creating a Complex View

- Create a complex view that contains group functions to display values from two tables.

```
SQL> CREATE VIEW      dept_sum_vu
      2          (name, minsal, maxsal, avgsal)
      3  AS SELECT      d.dname, MIN(e.sal), MAX(e.sal),
      4                  AVG(e.sal)
      5  FROM          emp e, dept d
      6  WHERE         e.deptno = d.deptno
      7  GROUP BY      d.dname;
View created.
```

Rules for Performing DML Operations on a View

- You can perform DML operations on simple views.
- You cannot remove a row if the view contains the following:
 - Group functions
 - A GROUP BY clause
 - The DISTINCT keyword
- You cannot add data if:
 - There are NOT NULL columns in the base tables that are not selected by the view
 - Columns defined by expressions

Using the WITH CHECK OPTION Clause

- You can ensure that DML on the view stays within the domain of the view by using the WITH CHECK OPTION clause.

```
SQL> CREATE OR REPLACE VIEW empvu20
  2  AS SELECT      *
  3  FROM          emp
  4  WHERE         deptno = 20
  5  WITH CHECK OPTION;
```

View created.

- Any attempt to change the department number for any row in the view will fail because it violates the WITH CHECK OPTION constraint.

Denying DML Operations

- You can ensure that no DML operations occur by adding the WITH READ ONLY option to your view definition.

```
SQL> CREATE OR REPLACE VIEW empvu10
  2      (employee_number, employee_name, job_title)
  3  AS SELECT          empno, ename, job
  4  FROM            emp
  5 WHERE          deptno = 10
  6 WITH READ ONLY;
```

View created.

- Any attempt to perform a DML on any row in the view will result in Oracle Server error.

Removing a View

- Remove a view without losing data because a view is based on underlying tables in the database.

```
DROP VIEW view;
```

```
SQL> DROP VIEW empvu10;
```

```
View dropped.
```



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



(SQL)

Other Database Objects

Asif Sohail

**University of the Punjab
Punjab University College of Information Technology (PUCIT)**

Objectives

- After completing this lesson, you should be able to do the following:
 - Describe some database objects and their uses
 - Create, maintain, and use sequences
 - Create and maintain indexes
 - Create private and public synonyms

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	Generates primary key values
Index	Improves the performance of some queries
Synonym	Alternative name for an object

What Is a Sequence?

- Automatically generates unique numbers
- Is a sharable object
- It can have max. value up to 28 digits.
- Is typically used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory

The CREATE SEQUENCE Statement

- Define a sequence to generate sequential numbers automatically.

```
CREATE SEQUENCE sequence
    [INCREMENT BY n]
    [START WITH n]
    [ {MAXVALUE n | NOMAXVALUE} ]
    [ {MINVALUE n | NOMINVALUE} ]
    [ {CYCLE | NOCYCLE} ]
    [ {CACHE n | NOCACHE} ] ;
```

The CREATE SEQUENCE Options

- **START WITH:** The starting value of the sequence. It can be of 28 or lesser digits.
- **MAXVALUE:** The maximum value to be generated by the sequence. Its default value is $10^{27} - 1$ for an ascending sequence and 1 for a descending sequence.
- **MINVALUE:** The minimum value to be generated by the sequence. Its default value is 1 for an ascending sequence and $10^{27} - 1$ for a descending sequence.
- **CYCLE / NOCYCLE:** It indicates whether the sequence will continue upon reaching its max/min value or not.
- **CACHE:** It is used for faster access. Its minimum value is 2 and default value is 20.

Creating a Sequence

- Create a sequence named DEPT_DEPTNO to be used for the primary key of the DEPT table.
- Do not use the CYCLE option.

```
SQL> CREATE SEQUENCE dept_deptno
  2      INCREMENT BY 1
  3      START WITH 91
  4      MAXVALUE 100
  5      NOCACHE
  6      NOCYCLE;
Sequence created.
```

Confirming Sequences

- Verify your sequence values in the USER_SEQUENCES data dictionary table.

```
SQL> SELECT    sequence_name, min_value, max_value,  
2          increment_by, last_number  
3  FROM      user_sequences;
```

- The LAST_NUMBER column displays the next available sequence number.

NEXTVAL and CURRVAL Pseudocolumns

- NEXTVAL returns the next available sequence value.
- It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

Using a Sequence

- Insert a new department named “MARKETING”.

```
SQL> INSERT INTO      dept(deptno, dname, loc)
   2  VALUES            (dept_deptno.NEXTVAL, 'MARKETING') ;
1 row created.
```

- View the current value for the DEPT_DEPTNO sequence

```
SQL> SELECT  dept_deptno.CURRVAL
   2  FROM    dual;
```

- Gaps in sequence values can occur when:
 - A rollback occurs
 - The system crashes
 - A sequence is used in another table

Modifying a Sequence

- Change the increment value, maximum value, minimum value, cycle option, or cache option.

```
SQL> ALTER SEQUENCE dept_deptno
  2      INCREMENT BY 1
  3      MAXVALUE 999999
  4      NOCACHE
  5      NOCYCLE;
```

Sequence altered.

Removing a Sequence

- Remove a sequence from the data dictionary by using the DROP SEQUENCE statement.
- Once removed, the sequence can no longer be referenced.

```
SQL> DROP SEQUENCE dept_deptno;  
Sequence dropped.
```

What Is an Index?

- Is used by the Oracle Server to speed up the retrieval of rows by using a pointer
- Can reduce disk I/O by using rapid path access method to locate the data quickly
- Is independent of the table it indexes
- Is used and maintained automatically by the Oracle Server

How Are Indexes Created?

- **Automatically:** A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a table definition.
- **Manually:** Users can create nonunique indexes on columns to speed up access time to the rows.

Creating an Index

- Create an index on one or more columns.

```
CREATE [UNIQUE] INDEX index
ON table (column[, column]...) ;
```

- Improve the speed of query access on the ENAME column in the EMP table.

```
SQL> CREATE INDEX      emp_ename_idx
      2 ON                  emp(ename);
Index created.
```

When to Create an Index

- The column is used frequently in the WHERE clause (secondary key) or in a join condition.
- A frequently accessed foreign key.
- The columns frequently involved in ORDER BY and GROUP BY clause.
- The column contains a wide range of values.
- The column contains a large number of null values.
- One or more columns are frequently used together in a WHERE clause or a join condition.
- The table is large and most queries are expected to retrieve less than 2–4% of the rows.

When Not to Create an Index

- The table is small.
- The columns are not often used as a condition in the query.
- Most queries are expected to retrieve more than 2–4% of the rows.
- The table or column is updated frequently.
- A column consisting of long character strings.

Confirming Indexes

- The USER_INDEXES data dictionary view contains the name of the index and its uniqueness.
- The USER_IND_COLUMNS view contains the index name, the table name, and the column name.

```
SQL> SELECT    ic.index_name, ic.column_name,
  2          ic.column_position col_pos, ix.uniqueness
  3  FROM      user_indexes ix, user_ind_columns ic
  4 WHERE     ic.index_name = ix.index_name
  5 AND       ic.table_name = 'EMP' ;
```

Removing an Index

- Remove an index from the data dictionary.

```
SQL> DROP INDEX index;
```

- Remove the EMP_ENAME_IDX index from the data dictionary.

```
SQL> DROP INDEX emp_ename_idx;  
Index dropped.
```

- To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.

Synonyms

- Simplify access to objects by creating a synonym (another name for an object).
 - Refer to a table owned by another user.
 - Shorten lengthy object names.

```
CREATE [PUBLIC] SYNONYM synonym
FOR      object;
```

Creating and Removing Synonyms

- Create a shortened name for the DEPT_SUM_VU view.

```
SQL> CREATE SYNONYM d_sum  
2 FOR dept_sum_vu;
```

Synonym Created.

- Drop a synonym.

```
SQL> DROP SYNONYM d_sum;
```

Synonym dropped.



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk



(SQL)

Controlling User Access

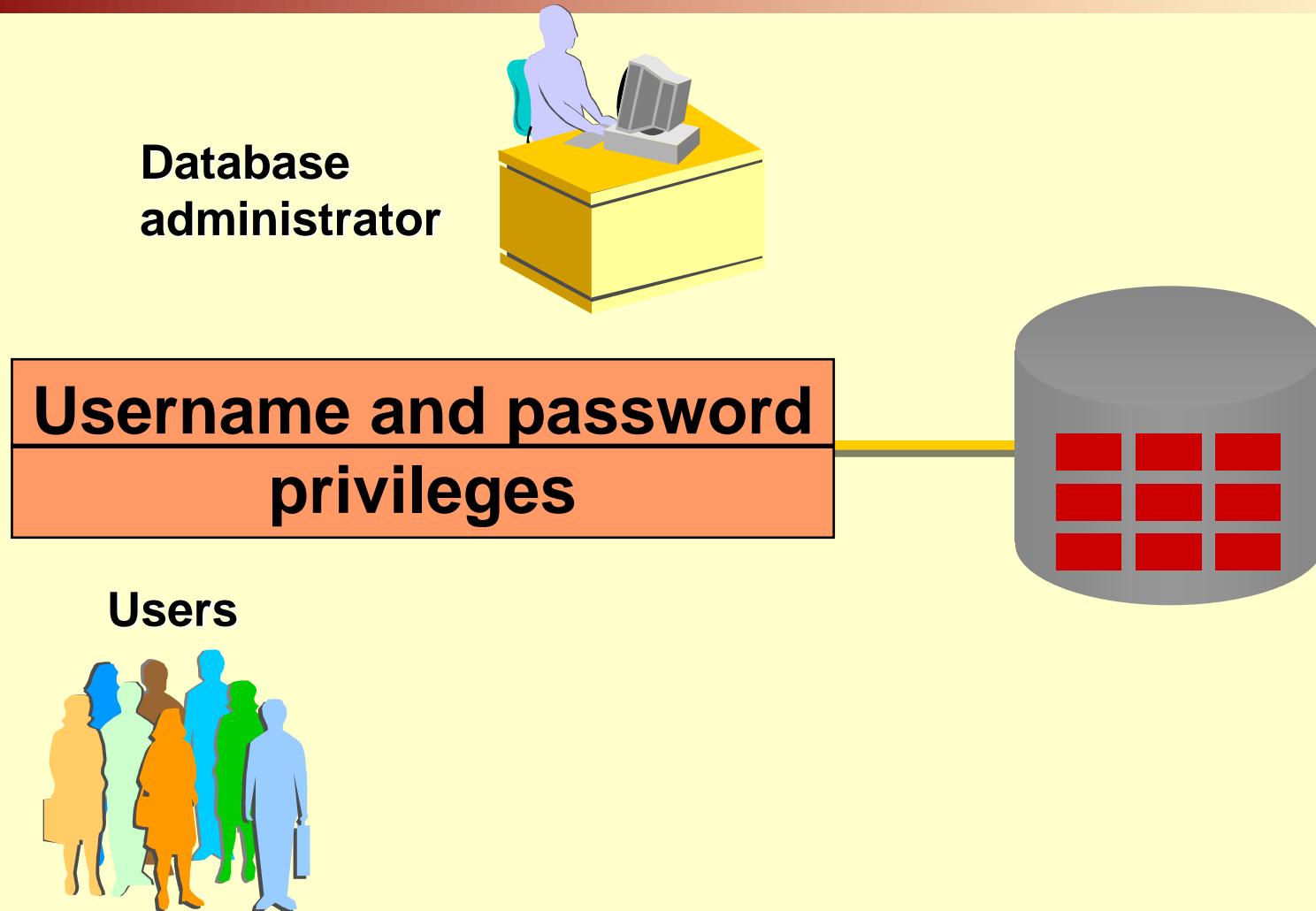
Asif Sohail

**University of the Punjab
Punjab University College of Information Technology (PUCIT)**

Objectives

- After completing this lesson, you should be able to do the following:
 - Create users
 - Create roles to ease setup and maintenance of the security model
 - Use the GRANT and REVOKE statements to grant and revoke object privileges

Controlling User Access



Privileges

- Database security:
 - System security
 - Data security
- System privileges: Gain access to the database
- Object privileges: Manipulate the content of the database objects
- More than 80 privileges are available.
- The DBA has high-level system privileges:
 - Create new users
 - Remove users
 - Remove tables
 - Back up tables

Creating Users

- The DBA creates users by using the CREATE USER statement.

```
CREATE USER      username
IDENTIFIED BY   password;
```

```
SQL> CREATE  USER  scott
  2 IDENTIFIED BY tiger;
User created.
```

- To log in with the new user:

```
CONNECT;
```

Changing Your Password

- The DBA creates your user account and initializes your password.
- You can change your password by using the ALTER USER statement.

```
SQL> ALTER USER scott  
2          IDENTIFIED BY lion;  
User altered.
```

Viewing all the Users

- Use the Data Dictionary view **ALL_USERS**.

```
SQL> DESC ALL_USERS;
```

Name	Null?	Type
USERNAME	NOT NULL	VARCHAR2 (30)
USER_ID	NOT NULL	NUMBER
CREATED	NOT NULL	DATE

- To get a list of all the users:

```
SQL> SELECT * FROM ALL_USERS;
```

- To get a list of all the users along with their privileges:

```
SQL> SELECT * FROM DBA_ROLE_PRIVS;
```

User System Privileges

- Once a user is created, the DBA can grant specific system privileges to a user.

```
GRANT privilege [, privilege...]  
TO user [, user...];
```

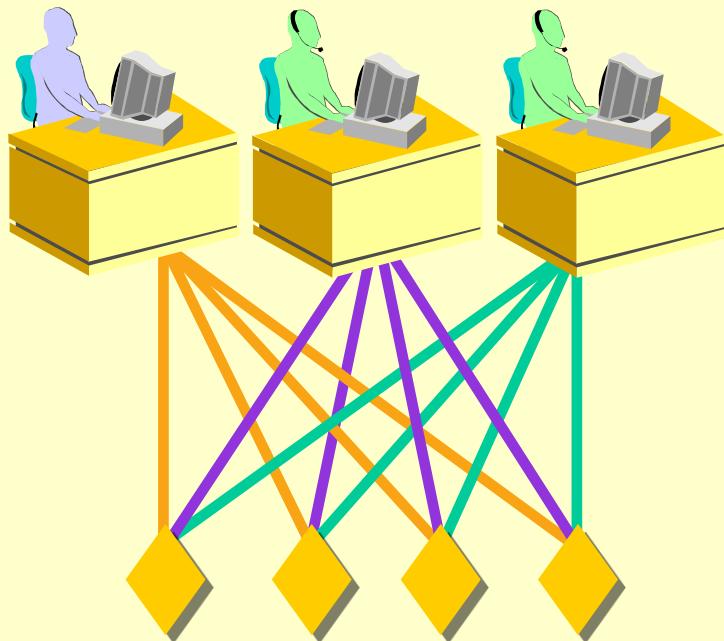
- An application developer may have the following system privileges:
 - CREATE TABLE
 - CREATE SEQUENCE
 - CREATE VIEW
 - CREATE INDEX
 - CREATE PROCEDURE

Granting System Privileges

- The DBA can grant a user specific system privileges.

```
SQL> GRANT    create table, create sequence, create view  
  2  TO        scott;  
Grant succeeded.
```

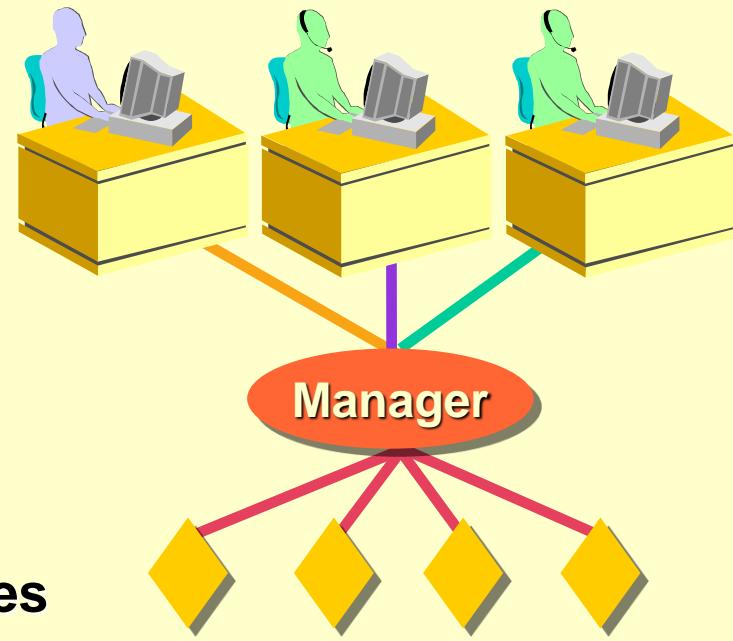
What Is a Role?



**Allocating privileges
without a role**

Users

Privileges



**Allocating privileges
with a role**

What Is a Role?

- A role is a privilege or set of privileges that allows a user to perform certain functions in the database.
- To grant a role to a user, use the following syntax:

```
SQL> GRANT role TO USER  
2          [WITH ADMIN OPTION] ;  
Grant succeeded.
```

- If WITH ADMIN OPTION is used, that user can then grant roles to other users.

Creating and Granting Privileges to a Role

```
SQL> CREATE ROLE manager;  
Role created.
```

```
SQL> GRANT create table, create view  
  2      to manager;  
Grant succeeded.
```

```
SQL> GRANT manager to BLAKE, CLARK;  
Grant succeeded.
```

Standard Role Names

- Oracle lets you register as one of three roles:
 - a) Connect
 - b) Resource
 - c) DBA (or database administrator)
- These three roles have varying degrees of privileges.

Standard Role Names

a) The Connect Role

- The Connect role can be thought of as the entry-level role.
- A user who has been granted Connect role access can be granted various privileges that allow him or her to do something with a database.
- The Connect role enables the user to select, insert, update, and delete records from tables belonging to other users (after the appropriate permissions have been granted).

Standard Role Names

b) The Resource Role

- The Resource role gives the user more access to Oracle databases.
- The user can also create tables, views, sequences, and synonyms.
- In addition to the permissions that can be granted to the Connect role, Resource roles can also be granted permission to create procedures, triggers, and indexes.

Standard Role Names

c) The DBA Role

- The DBA role includes all privileges.
- Users with this role are able to do essentially anything they want to the database system.

Granting and Revoking a Role

```
SQL> GRANT role TO user;
```

```
SQL> GRANT connect TO manager;
```

```
SQL> REVOKE role FROM user;
```

```
SQL> REVOKE connect FROM manager;
```

```
SQL> REVOKE create table FROM manager;
```

Object Privileges

- After you decide which roles to grant your users, your next step is deciding which permissions or privileges these users will have on database objects.
- If you actually create an object, you can grant privileges on that object to other users.
- Object privileges vary from object to object.
- An owner has all the privileges on the object.

```
GRANT      {object_priv | ALL} [(columns)]  
ON         object  
TO         {user|role|PUBLIC}  
[WITH GRANT OPTION];
```

Object Privileges

Object Privilege	Table	View	Sequence	Procedure
•ALTER	Ö		Ö	
•DELETE	Ö	Ö		
•EXECUTE				Ö
•INDEX	Ö			
•INSERT	Ö	Ö		
•REFERENCES	Ö			
•SELECT	Ö	Ö	Ö	
•UPDATE	Ö	Ö		

Granting Object Privileges

- Grant query privileges on the EMP table.

```
SQL> GRANT      select  
  2  ON          emp  
  3  TO          boota,bala;
```

Grant succeeded.

- Grant all privileges on the EMP table.

```
SQL> GRANT      ALL ON emp  
  3  TO          boss;
```

Grant succeeded.

- Grant privileges to update specific columns to users and roles.

```
SQL> GRANT      update (dname, loc)  
  2  ON          dept  
  3  TO          scott, manager;
```

Grant succeeded.

Using WITH GRANT OPTION and PUBLIC Keywords

- Give a user authority to pass along the privileges.

```
SQL> GRANT      select, insert  
  2  ON          dept  
  3  TO          boss  
  4  WITH GRANT OPTION;
```

Grant succeeded.

- Now the user / role boss can pass on the above granted privileges to other users/roles in the following way:..

```
SQL> GRANT      select  
  2  ON          scott.dept  
  3  TO          PUBLIC;
```

Grant succeeded.

Using the objects of other users

Qualifying the granted object

- Let A is a table name create by user X and X has granted SELECT privilege on A to a user Y.
- User Y can SELECT rows from table A using:
- **SELECT * FROM X.A;**
- If a user wants to omit the qualifier name, he/she can create a synonym for that object.
- **CREATE SYNONYM A for X.A;**

Confirming Privileges Granted

Data Dictionary Table	Description
ROLE_SYS_PRIVS	System privileges granted to roles
ROLE_TAB_PRIVS	Table privileges granted to roles
USER_ROLE_PRIVS	Roles accessible by the user
USER_TAB_PRIVS_MADE	Object privileges granted on the user's objects
USER_TAB_PRIVS_REC'D	Object privileges granted to the user
USER_COL_PRIVS_MADE	Object privileges granted on the columns of the user's objects
USER_COL_PRIVS_REC'D	Object privileges granted to the user on specific columns

How to Revoke Object Privileges

- You use the REVOKE statement to revoke privileges granted to other users.
- Privileges granted to others through the WITH GRANT OPTION will also be revoked.

```
REVOKE {privilege [, privilege...] | ALL}
ON      object
FROM    {user[, user...] | role | PUBLIC};
```

Revoking Object Privileges

- As user Alice, revoke the SELECT and INSERT privileges given to user Scott on the DEPT table.

```
SQL> REVOKE    select, insert  
  2  ON        dept  
  3  FROM      scott;  
Revoke succeeded.
```

Summary

•Statement	Action
•CREATE USER	Allows the DBA to create a user
•GRANT	Allows the user to give other users privileges to access the user's objects
•CREATE ROLE	Allows the DBA to create a collection of privileges
•ALTER USER	Allows users to change their password
•REVOKE	Removes privileges on an object from users

Practice Overview

- Granting other users privileges to your table
- Modifying another user's table through the privileges granted to you
- Creating a synonym
- Querying the data dictionary views related to privileges



Thank you for your attention.

Asif Sohail

Assistant Professor

University of the Punjab

Punjab University College of Information Technology (PUCIT)

Allama Iqbal (Old) Campus, Anarkali

Lahore, Pakistan

Tel: +92-(0)42-111-923-923 Ext. 154

E-mail: asif@pucit.edu.pk