Relational Database Terminology

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	90
101	Neena	Kochhar	17000	(null)	90
102	Lex	De Haan	17000	(null)	90
103	Alexander	Hunold	9000	(null)	60
104	Bruce	Ernst	6000	(null)	60
107	Diana	Lorentz	4200	(null)	60
124	Kevin	Mourgos	5800	(null)	50
141	Trenna	Rajs	3500	(null)	50
142	Curtis	Davies	3100	(null)	50
143	Randall	Matos	2600	(null)	50
144	Peter	Vargas	2500	(null)	50
149	Eleni	Zlotkey	10500	0.2	80
174	Ellen	Abel	11000	0.3	80
176	Jonathon	Taylor	8600	0.2	80
178	Kimberely	Grant	7000	0.15	(null)
200	Jennifer	Whalen	4400	(null)	10
201	Michael	Hartstein	13000	(null)	20
202	Pat	Fay	6000	(null)	20
205	Shelley	Higgins	12000	(null)	110
206	William	Gietz	8300	(null)	110

2 **3** **4** **5** **6** **1**

Oracle University and BUSINESS SUPPORT LTDA use only

SQL Statements

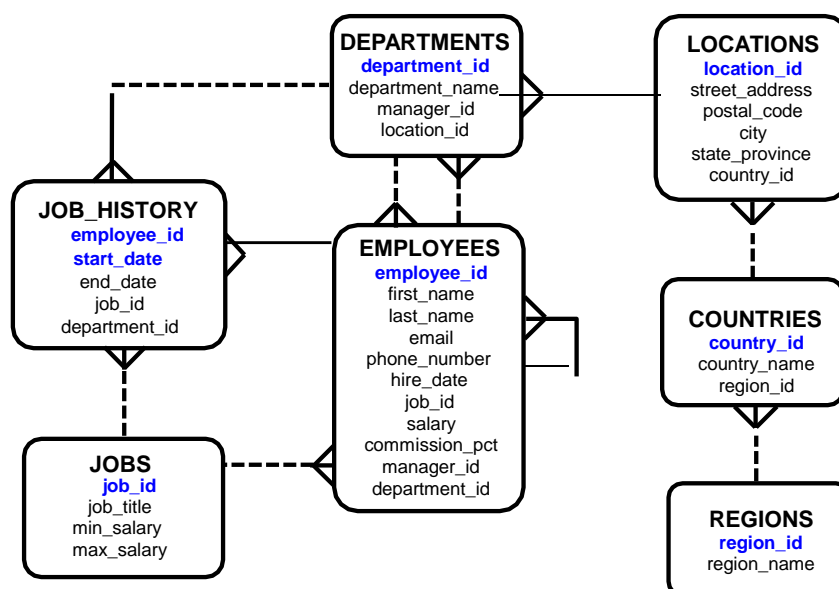|	
SELECT INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Data definition language (DDL)
GRANT REVOKE	Data control language (DCL)
COMMIT ROLLBACK SAVEPOINT	Transaction control

The `Human Resources (HR)` Schema

DEPARTMENTS
department_id
department_name
manager_id
location_id

LOCATIONS
location_id
street_address
postal_code
city
state_province
country_id

JOB_HISTORY
employee_id
start_date
end_date
job_id
department_id

EMPLOYEES
employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
commission_pct
manager_id
department_id

COUNTRIES
country_id
country_name
region_id

JOBS
job_id
job_title
min_salary
max_salary

REGIONS
region_id
region_name

Oracle University and BUSINESS SUPPORT LTDA use only

The `Human Resources (HR)` Schema Description

Retrieving Data Using
the SQL SELECT Statement

ORACLE

Capabilities of SQL `SELECT` Statements

Projection

Table 1

Selection

Table 1

Table 1

Join

Table 2

Basic SELECT Statement

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM    table;
```

- SELECT identifies the columns to be displayed.
- FROM identifies the table containing those columns.

Selecting All Columns

```
SELECT *
FROM   departments;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Selecting Specific Columns

```
SELECT department_id, location_id
FROM   departments;
```

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

ORACLE

Copyright © 2009, Oracle. All rights reserved.

...

Writing SQL Statements

- SQL statements are not case-sensitive.
- SQL statements can be entered on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.
- In SQL Developer, SQL statements can optionally be terminated by a semicolon (;). Semicolons are required when you execute multiple SQL statements.
- In SQL*Plus, you are required to end each SQL statement with a semicolon (;).

Column Heading Defaults

- SQL Developer:
  - Default heading alignment: Left-aligned
  - Default heading display: Uppercase
- SQL*Plus:
  - Character and Date column headings are left-aligned.
  - Number column headings are right-aligned.
  - Default heading display: Uppercase

Arithmetic Expressions

Create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

ORACLE

Operator Precedence

```
SELECT last_name, salary, 12*salary+100
FROM    employees;
```

(1)

	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100

...

```
SELECT last_name, salary, 12*(salary+100)
FROM    employees;
```

(2)

	LAST_NAME	SALARY	12*(SALARY+100)
1	King	24000	289200
2	Kochhar	17000	205200
3	De Haan	17000	205200

...

ORACLE

Oracle Database 11*g*: SQL Fundamentals I 1 - 13

Defining a Null Value

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

```
SELECT last_name, job_id, salary,
FROM    employees;
```

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)

. . .

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2

. . .

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
19	Higgins	AC_MGR	12000	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

ORACLE

Defining a Column Alias

A column alias:

- Renames a column heading

- Is useful with calculations

- Immediately follows the column name (There can also be the optional `AS` keyword between the column name and alias.)

- Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive

Oracle University and BUSINESS SUPPORT LTDA use only

Using Column Aliases

```
SELECT last_name AS name , commission_pct comm
FROM   employees;
```

NAME	COMM
1 King	(null)
2 Kochhar	(null)
3 De Haan	(null)

...

```
SELECT last_name "Name" , salary*12 "Annual Salary"
FROM   employees;
```

Name	Annual Salary
1 King	288000
2 Kochhar	204000
3 De Haan	204000

...

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Lesson Agenda

- Basic SELECT Statement
- Arithmetic Expressions and NULL values in SELECT statement
- Column Aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the DISTINCT keyword
- DESCRIBE command

Concatenation Operator

A concatenation operator:

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

```
SELECT    last_name||job_id AS "Employees"
FROM      employees;
```

	Employees
1	AbelSA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG
5	FayMK_REP

...

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Using Literal Character Strings

```
SELECT last_name ||' is a '||job_id
       AS "Employee Details"
FROM   employees;
```

	Employee Details
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP

...

18	Vargas is a ST_CLERK
19	Whalen is a AD_ASST
20	Zlotkey is a SA_MAN

ORACLE

Using Literal Character Strings

The example in the slide displays the last names and job codes of all employees. The column has the heading Employee Details. Note the spaces between the single quotation marks in the SELECT statement. The spaces improve the readability of the output.

In the following example, the last name and salary for each employee are concatenated with a literal, to give the returned rows more meaning:

```
SELECT last_name ||': 1 Month salary = '||salary Monthly
FROM   employees;
```

	MONTHLY
1	King: 1 Month salary = 24000
2	Kochhar: 1 Month salary = 17000
3	De Haan: 1 Month salary = 17000
4	Hunold: 1 Month salary = 9000
5	Ernst: 1 Month salary = 6000
6	Lorentz: 1 Month salary = 4200
7	Mourgos: 1 Month salary = 5800
8	Rajs: 1 Month salary = 3500

Alternative Quote (q) Operator

- Specify your own quotation mark delimiter.
- Select any delimiter.
- Increase readability and usability.

```
SELECT department_name ||
       || manager_id
       AS "Department and Manager"
FROM departments;
```

	Department and Manager
1	Administration Department's Manager Id:200
2	Marketing Department's Manager Id:201
3	Shipping Department's Manager Id:124
4	IT Department's Manager Id:103
5	Sales Department's Manager Id:149
6	Executive Department's Manager Id:100
7	Accounting Department's Manager Id:205
8	Contracting Department's Manager Id:

ORACLE

Duplicate Rows

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

```
SELECT department_id
FROM   employees;
```
1

	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60

...

```
SELECT DISTINCT department_id
FROM   employees;
```
2

	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110

...

ORACLE

Lesson Agenda

- Basic `SELECT` statement
- Arithmetic expressions and `NULL` values in the `SELECT` statement
- Column aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

Oracle University and BUSINESS SUPPORT LTDA use only

Using the DESCRIBE Command

```
DESCRIBE employees
```

```
DESCRIBE employees
Name                            Null      Type
------------------------------- --------  --------------------
EMPLOYEE_ID                     NOT NULL  NUMBER(6)
FIRST_NAME                                VARCHAR2(20)
LAST_NAME                       NOT NULL  VARCHAR2(25)
EMAIL                           NOT NULL  VARCHAR2(25)
PHONE_NUMBER                              VARCHAR2(20)
HIRE_DATE                       NOT NULL  DATE
JOB_ID                          NOT NULL  VARCHAR2(10)
SALARY                                    NUMBER(8,2)
COMMISSION_PCT                            NUMBER(2,2)
MANAGER_ID                                NUMBER(6)
DEPARTMENT_ID                             NUMBER(4)

11 rows selected
```

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Quiz

Identify the SELECT statements that execute successfully.

1.
```
SELECT first_name, last_name, job_id, salary*12
 AS Yearly Sal
FROM    employees;
```

2.
```
SELECT first_name, last_name, job_id, salary*12
 yearly sal
FROM    employees;
```

3.
```
SELECT first_name, last_name, job_id, salary AS
 yearly sal
FROM    employees;
```

4.
```
SELECT first_name+last_name AS name, job_Id,
 salary*12 yearly sal
FROM    employees;
```

ORACLE

Answer: 2, 3

Restricting and Sorting Data

2

Oracle University and BUSINESS SUPPORT LTDA use only

Limiting Rows Using a Selection

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90
4	103	Hunold	IT_PROG	60
5	104	Ernst	IT_PROG	60
6	107	Lorentz	IT_PROG	60

…

"retrieve all employees in department 90"

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

ORACLE

Limiting the Rows That Are Selected

- Restrict the rows that are returned by using the `WHERE` clause:

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

- The `WHERE` clause follows the `FROM` clause.

ORACLE

Using the `WHERE` Clause

```
SELECT employee_id, last_name, job_id, department_id
FROM    employees
WHERE   department_id = 90 ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

Character Strings and Dates

- Character strings and date values are enclosed with single quotation marks.
- Character values are case-sensitive and date values are format-sensitive.
- The default date display format is `DD-MON-RR`.

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name =            ;
```

```
SELECT last_name
FROM   employees
WHERE  hire_date = '17-FEB-96' ;
```

Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN ...AND...	Between two values (inclusive)
IN(set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

ORACLE

Using Comparison Operators

```
SELECT last_name, salary
FROM   employees
WHERE  salary <= 3000 ;
```

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

Range Conditions Using the BETWEEN Operator

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

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500 ;
```

Lower limit   **Upper limit**

	LAST_NAME		SALARY
1	Rajs		3500
2	Davies		3100
3	Matos		2600
4	Vargas		2500

ORACLE

Membership Condition Using the **IN** Operator

Use the **IN** operator to test for values in a list:

```
SELECT  employee_id, last_name, salary, manager_id
FROM    employees
WHERE   manager_id                         ;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12000	101
8	202	Fay	6000	201

ORACLE

```
SELECT employee_id, manager_id, department_id
  FROM   employees
  WHERE last_name IN ('Hartstein', 'Vargas');
```

Pattern Matching 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.

```
SELECT    first_name
FROM      employees
WHERE     first_name LIKE 'S%' ;
```

Symbol	Description
%	Represents any sequence of zero or more characters
_	Represents any single character

Combining Wildcard Characters

- You can combine the two wildcard characters (%, _) with literal characters for pattern matching:

```
SELECT last_name
FROM    employees
WHERE   last_name LIKE '_o%' ;
```

	LAST_NAME
1	Kochhar
2	Lorentz
3	Mourgos

- You can use the ESCAPE identifier to search for the actual % and _ symbols.

ORACLE

Using the **NULL** Conditions

Test for nulls with the IS NULL operator.

```
SELECT last_name, manager_id
FROM    employees
WHERE   manager_id IS NULL ;
```

	LAST_NAME	MANAGER_ID
1	King	(null)

ORACLE

Defining Conditions Using the Logical Operators

Operator	Meaning
AND	Returns TRUE if *both* component conditions are true
OR	Returns TRUE if *either* component condition is true
NOT	Returns TRUE if the condition is false

Using the AND Operator

AND requires both the component conditions to be true:

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
AND    job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	149	Zlotkey	SA_MAN	10500
2	201	Hartstein	MK_MAN	13000

ORACLE

Using the OR Operator

OR requires either component condition to be true:

```
SELECT employee_id, last_name, job_id, salary
FROM    employees
WHERE   salary >= 10000
OR      job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	100	King	AD_PRES	24000
2	101	Kochhar	AD_VP	17000
3	102	De Haan	AD_VP	17000
4	124	Mourgos	ST_MAN	5800
5	149	Zlotkey	SA_MAN	10500
6	174	Abel	SA_REP	11000
7	201	Hartstein	MK_MAN	13000
8	205	Higgins	AC_MGR	12000

ORACLE

Using the NOT Operator

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
       NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP') ;
```

	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

ORACLE

Oracle Database 11*g*: SQL Fundamentals I 2 - 18

Lesson Agenda

- Limiting rows with:
  - The `WHERE` clause
  - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
  - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- `DEFINE` and `VERIFY` commands

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Rules of Precedence

Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	`IS [NOT] NULL, LIKE, [NOT] IN`
5	`[NOT] BETWEEN`
6	Not equal to
7	`NOT` logical condition
8	`AND` logical condition
9	`OR` logical condition

You can use parentheses to override rules of precedence.

ORACLE

Rules of Precedence

```
SELECT last_name, job_id, salary
FROM    employees
WHERE   job_id = 'SA_REP'
OR      job_id = 'AD_PRES'
AND     salary > 15000;
```

①

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000
2	Abel	SA_REP	11000
3	Taylor	SA_REP	8600
4	Grant	SA_REP	7000

```
SELECT last_name, job_id, salary
FROM    employees
WHERE   (job_id = 'SA_REP'
OR      job_id = 'AD_PRES')
AND     salary > 15000;
```

②

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000

ORACLE

Using the ORDER BY Clause

- Sort retrieved rows with the ORDER BY clause:
  - ASC: Ascending order, default
  - DESC: Descending order
- The ORDER BY clause comes last in the SELECT statement:

```
SELECT    last_name, job_id, department_id, hire_date
FROM      employees
ORDER BY hire_date ;
```

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	King	AD_PRES	90	17-JUN-87
2	Whalen	AD_ASST	10	17-SEP-87
3	Kochhar	AD_VP	90	21-SEP-89
4	Hunold	IT_PROG	60	03-JAN-90
5	Ernst	IT_PROG	60	21-MAY-91
6	De Haan	AD_VP	90	13-JAN-93

...

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Sorting

- Sorting in descending order:

```
SELECT    last_name, job_id, department_id, hire_date
FROM      employees
ORDER BY hire_date DESC ;
```
1

- Sorting by column alias:

```
SELECT employee_id, last_name, salary*12 annsal
FROM    employees
ORDER BY annsal ;
```
2

ORACLE

Sorting

- Sorting by using the column's numeric position:

```
SELECT    last_name, job_id, department_id, hire_date
FROM      employees
ORDER BY 3;
```
③

- Sorting by multiple columns:

```
SELECT last_name, department_id, salary
FROM    employees
ORDER BY department_id, salary DESC;
```
④

ORACLE

Substitution Variables

- Use substitution variables to:
  - Temporarily store values with single-ampersand (`&`) and double-ampersand (`&&`) substitution

- Use substitution variables to supplement the following:
  - `WHERE` conditions
  - `ORDER BY` clauses
  - Column expressions
  - Table names
  - Entire `SELECT` statements

Oracle University and BUSINESS SUPPORT LTDA use only
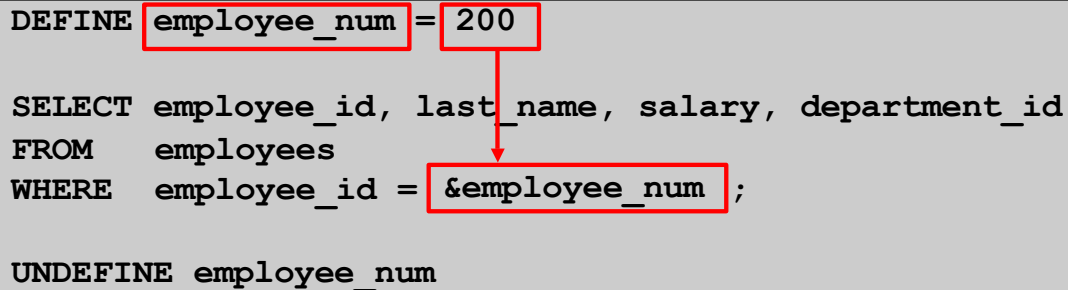
Using the DEFINE Command

- Use the `DEFINE` command to create and assign a value to a variable.
- Use the `UNDEFINE` command to remove a variable.

```
DEFINE employee_num = 200

SELECT employee_id, last_name, salary, department_id
FROM    employees
WHERE   employee_id = &employee_num ;

UNDEFINE employee_num
```

Using the DEFINE Command

The example shown creates a substitution variable for an employee number by using the `DEFINE` command. At run time, this displays the employee number, name, salary, and department number for that employee.

Because the variable is created using the SQL Developer `DEFINE` command, the user is not prompted to enter a value for the employee number. Instead, the defined variable value is automatically substituted in the `SELECT` statement.

The `EMPLOYEE_NUM` substitution variable is present in the session until the user undefines it or exits the SQL Developer session.

Quiz

Which of the following are valid operators for the `WHERE`
clause?
1. `>=`
2. `IS NULL`
3. `!=`
4. `IS LIKE`
5. `IN BETWEEN`
6. `<>`

Using Single-Row Functions to Customize Output

3

Oracle University and BUSINESS SUPPORT LTDA use only

SQL Functions

Input

Output

arg 1	
arg 2	

Function

Function performs action

Result value

arg n
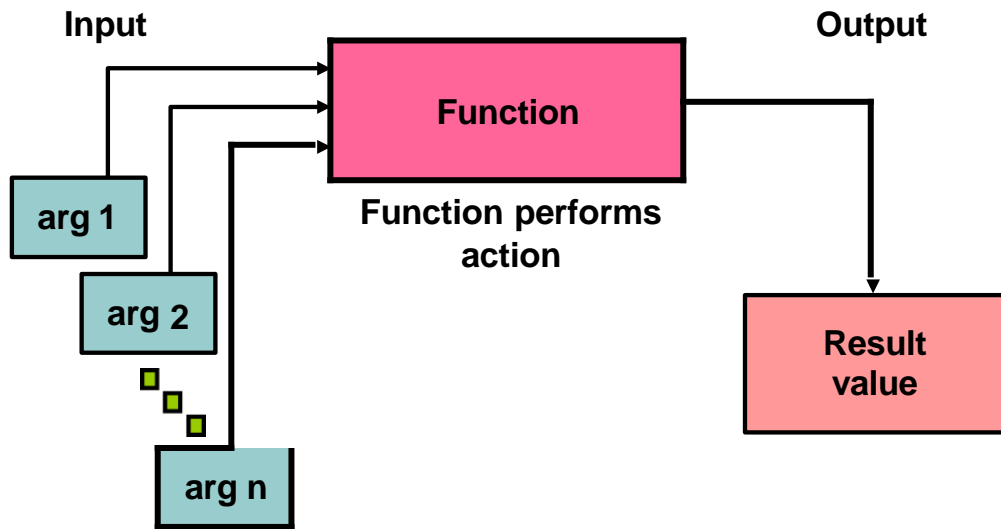
ORACLE

Two Types of SQL Functions
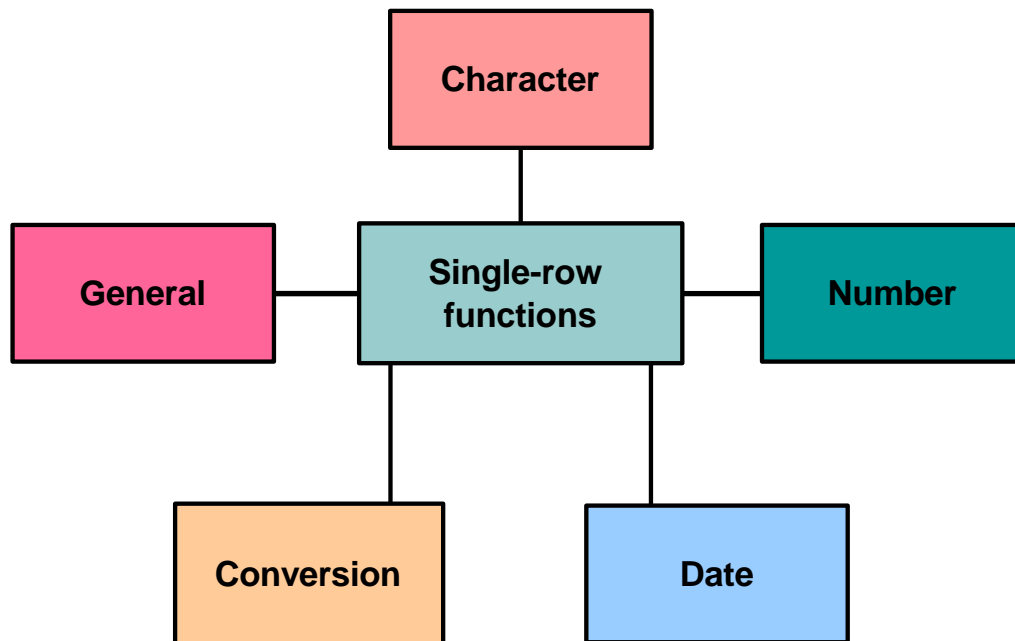
Single-Row Functions

Single-row functions:

- Manipulate data items
- Accept arguments and return one value
- Act on each row that is returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments that can be a column or an expression
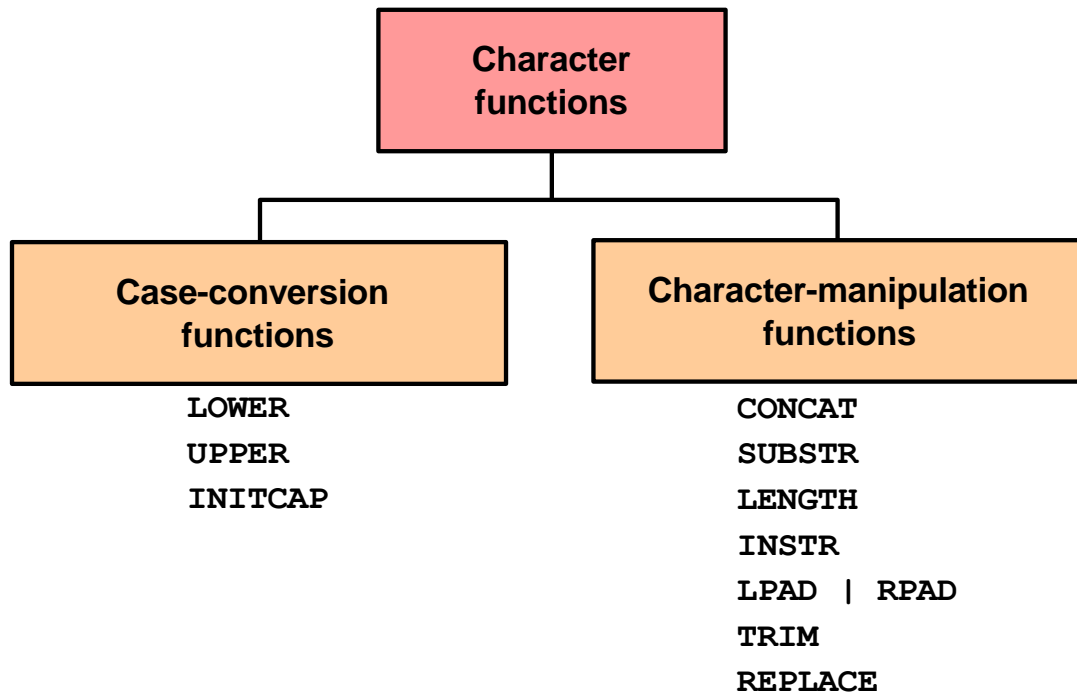
```
function_name [(arg1, arg2,...)]
```
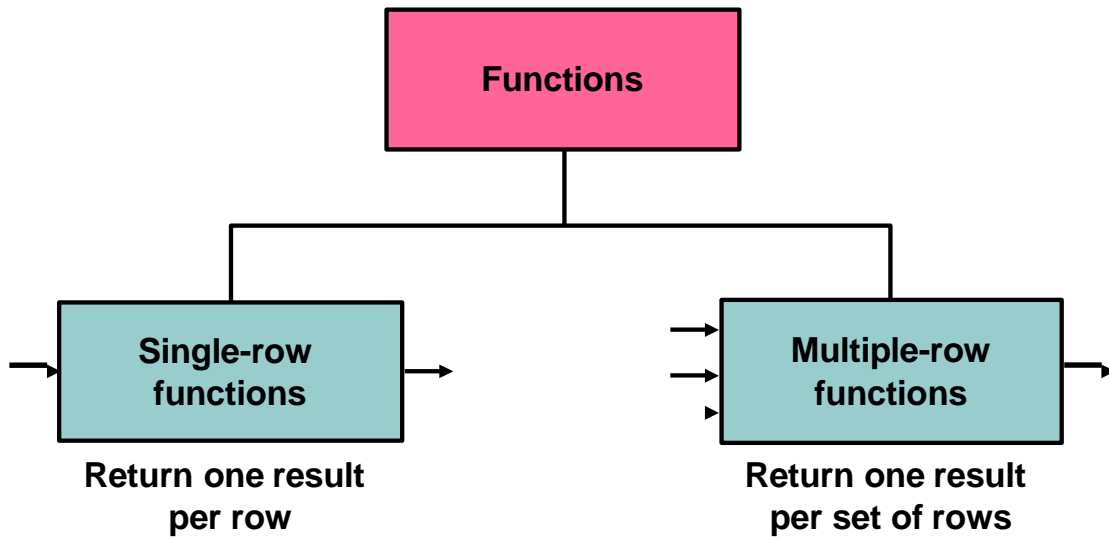
ORACLE

Single-Row Functions

```
                    ┌──────────────┐
                    │  Character   │
                    └──────┬───────┘
                           │
┌──────────────┐   ┌───────┴────────┐   ┌──────────────┐
│   General    │───│  Single-row    │───│    Number    │
└──────────────┘   │  functions     │   └──────────────┘
                   └───────┬────────┘
         ┌─────────────────┴─────────────────┐
         │                                   │
  ┌──────┴───────┐                    ┌───────┴──────┐
  │  Conversion  │                    │     Date     │
  └──────────────┘                    └──────────────┘
```

ORACLE

Character Functions


```
                        Character
                        functions

        Case-conversion          Character-manipulation
          functions                    functions

          LOWER                        CONCAT
          UPPER                        SUBSTR
          INITCAP                      LENGTH
                                       INSTR
                                       LPAD | RPAD
                                       TRIM
                                       REPLACE
```

ORACLE

Note: The functions discussed in this lesson are only some of the available functions.

Case-Conversion Functions

These functions convert the case for character strings:

Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

ORACLE

Using Case-Conversion Functions

Display the employee number, name, and department number for employee Higgins:

```
SELECT  employee_id, last_name, department_id
FROM    employees
WHERE   last_name = 'higgins';
```
`0 rows selected`

```
SELECT  employee_id, last_name, department_id
FROM    employees
WHERE   LOWER(last_name) = 'higgins';
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205 Higgins	110

ORACLE

Character-Manipulation Functions
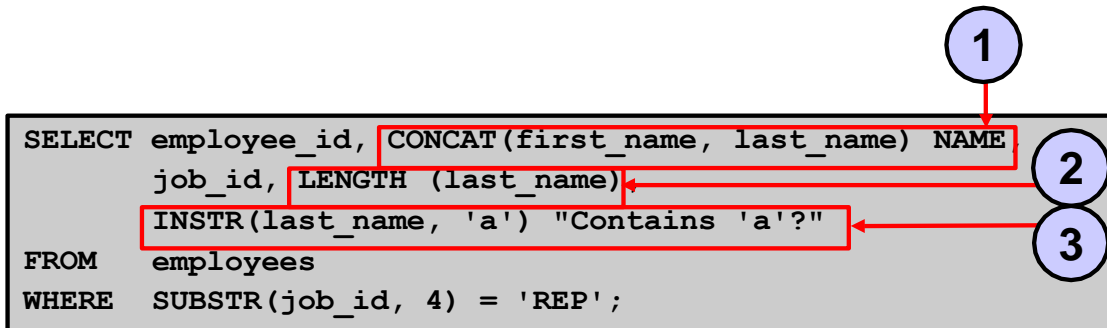
These functions manipulate character strings:

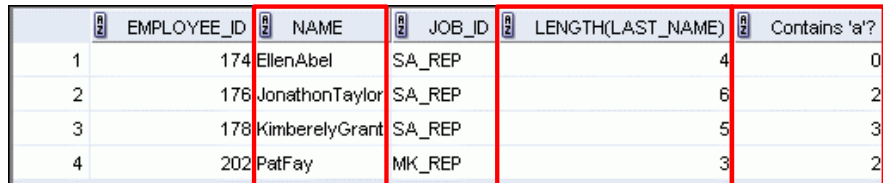| Function	Result
`CONCAT('Hello', 'World')`	`HelloWorld`
`SUBSTR('HelloWorld',1,5)`	`Hello`
`LENGTH('HelloWorld')`	`10`
`INSTR('HelloWorld', 'W')`	`6`
`LPAD(salary,10,'*')`	`*****24000`
`RPAD(salary, 10, '*')`	`24000*****`
`REPLACE` `('JACK and JUE','J','BL')`	`BLACK and BLUE`
`TRIM('H' FROM 'HelloWorld')`	`elloWorld`

ORACLE

Using the Character-Manipulation Functions

1

```
SELECT  employee_id, CONCAT(first_name, last_name) NAME,
        job_id, LENGTH (last_name),
        INSTR(last_name, 'a') "Contains 'a'?"
FROM    employees
WHERE   SUBSTR(job_id, 4) = 'REP';
```

2

3

	EMPLOYEE_ID	NAME	JOB_ID	LENGTH(LAST_NAME)	Contains 'a'?
1	174	EllenAbel	SA_REP	4	0
2	176	JonathonTaylor	SA_REP	6	2
3	178	KimberelyGrant	SA_REP	5	3
4	202	PatFay	MK_REP	3	2

1   **2**   **3**

ORACLE

Number Functions

- `ROUND`: Rounds value to a specified decimal
- `TRUNC`: Truncates value to a specified decimal
- `MOD`: Returns remainder of division

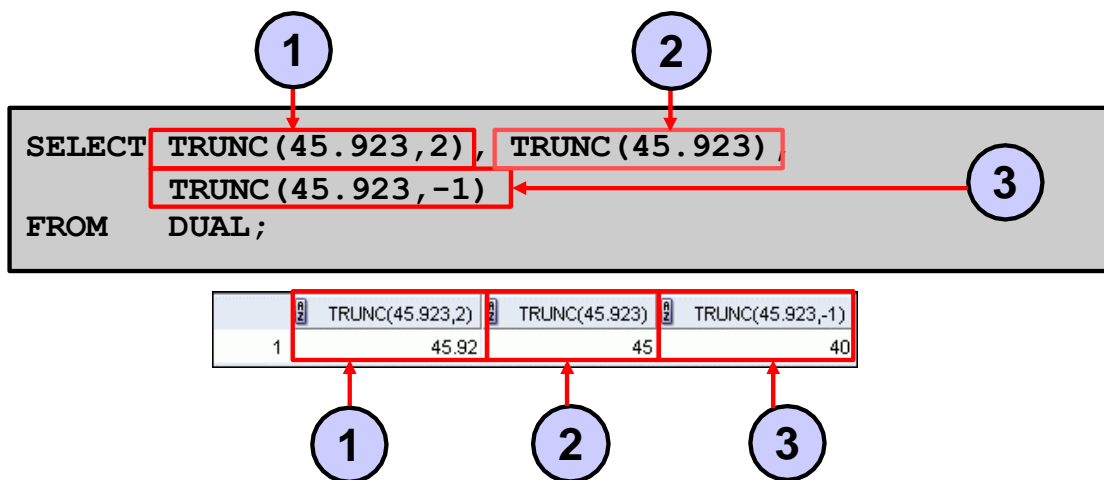| Function	Result
`ROUND(45.926, 2)`	`45.93`
`TRUNC(45.926, 2)`	`45.92`
`MOD(1600, 300)`	`100`

ORACLE

Using the ROUND Function


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

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

DUAL is a dummy table that you can use to view results from functions and calculations.

ORACLE

Using the TRUNC Function


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

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

Using the MOD Function

For all employees with the job title of Sales Representative, calculate the remainder of the salary after it is divided by 5,000.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM    employees
WHERE   job_id = 'SA_REP';
```

	LAST_NAME	SALARY	MOD(SALARY,5000)
1	Abel	11000	1000
2	Taylor	8600	3600
3	Grant	7000	2000

ORACLE

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- **Working with dates**
- Date functions

Working with Dates

- The Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.

- The default date display format is DD-MON-RR.
  - Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year
  - Enables you to store 20th-century dates in the 21st century in the same way

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '01-FEB-88';
```

	LAST_NAME	HIRE_DATE
1	King	17-JUN-87
2	Whalen	17-SEP-87

ORACLE

RR Date Format

Current Year	Specified Date	RR Format	YY Format
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		If the specified two-digit year is:	
		0–49	50–99
If two digits of the current year are:	0–49	The return date is in the current century	The return date is in the century before the current one
	50–99	The return date is in the century after the current one	The return date is in the current century

ORACLE

RR Date Format

The RR date format is similar to the YY element, but you can use it to specify different centuries. Use the RR date format element instead of YY so that the century of the return value varies according to the specified two-digit year and the last two digits of the current year. The table in the slide summarizes the behavior of the RR element.

Current Year	Given Date	Interpreted (RR)	Interpreted (YY)
1994	27-OCT-95	1995	1995
1994	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017

Using the SYSDATE Function

SYSDATE is a function that returns:

- Date
- Time

```
SELECT sysdate
FROM   dual;
```

	SYSDATE
1	31-MAY-07

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.

ORACLE

Operation	Result	Description
date + number	Date	Adds a number of days to a date
date – number	Date	Subtracts a number of days from a date
date – date	Number of days	Subtracts one date from another
date + number/24	Date	Adds a number of hours to a date

Using Arithmetic Operators
with Dates

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM    employees
WHERE   department_id = 90;
```

	LAST_NAME	WEEKS
1	King	1041.16823908730158730158730158730 1587302
2	Kochhar	923.02538194444444444444444444444 44444444
3	De Haan	750.16823908730158730158730158730 1587302

ORACLE

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- Working with dates
- Date functions

Oracle University and BUSINESS SUPPORT LTDA use only

Date-Manipulation Functions

Function	Result
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

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Using Date Functions

Function	Result
MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS ('31-JAN-96',1)	'29-FEB-96'
NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

ORACLE

Using ROUND and TRUNC Functions with Dates

Assume `SYSDATE = '25-JUL-03'`:

Function	Result
ROUND(SYSDATE,'MONTH')	01-AUG-03
ROUND(SYSDATE ,'YEAR')	01-JAN-04
TRUNC(SYSDATE ,'MONTH')	01-JUL-03
TRUNC(SYSDATE ,'YEAR')	01-JAN-03

ORACLE

Quiz

Which of the following statements are true about single-row functions?

1. Manipulate data items
2. Accept arguments and return one value per argument
3. Act on each row that is returned
4. Return one result per set of rows
5. May not modify the data type
6. Can be nested
7. Accept arguments that can be a column or an expression

Using Conversion Functions and Conditional Expressions

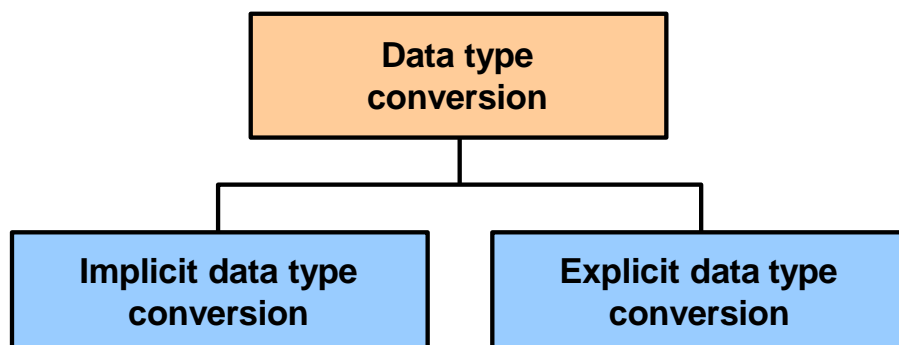Oracle University and BUSINESS SUPPORT LTDA use only

ORACLE

Lesson Agenda

- **Implicit and explicit data type conversion**
- `TO_CHAR, TO_DATE, TO_NUMBER` functions
- Nesting functions
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - `DECODE`

ORACLE

Conversion Functions

```
        ┌─────────────────┐
        │   Data type     │
        │   conversion    │
        └────────┬────────┘
         ┌───────┴───────┐
┌────────────────┐  ┌────────────────┐
│ Implicit data type │ Explicit data type │
│   conversion   │  │   conversion   │
└────────────────┘  └────────────────┘
```

Explicit Data Type Conversion

TO_NUMBER   TO_DATE

NUMBER   CHARACTER   DATE

TO_CHAR   TO_CHAR

ORACLE

Using the **TO_CHAR** Function with Dates

```
TO_CHAR(date, 'format_model')
```

The format model:

- Must be enclosed with single quotation marks
- Is case-sensitive
- Can include any valid date format element
- Has an `fm` element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

Elements of the Date Format Model

Element	Result
YYYY	Full year in numbers
YEAR	Year spelled out (in English)
MM	Two-digit value for the month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month

ORACLE

Elements of the Date Format Model

- Time elements format the time portion of the date:

HH24:MI:SS AM	15:45:32 PM

- Add character strings by enclosing them with double quotation marks:

DD "of" MONTH	12 of OCTOBER

- Number suffixes spell out numbers:

ddspth	fourteenth

Element	Description
AM or PM	Meridian indicator
A.M. or P.M.	Meridian indicator with periods
HH or HH12 or HH24	Hour of day, or hour (1–12), or hour (0–23)
MI	Minute (0–59)
SS	Second (0–59)
SSSSS	Seconds past midnight (0–86399)

Oracle University and BUSINESS SUPPORT LTDA use only

Using the `TO_CHAR` Function with Dates

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY')
       AS HIREDATE
FROM   employees;
```

	LAST_NAME	HIREDATE
1	King	17 June 1987
2	Kochhar	21 September 1989
3	De Haan	13 January 1993
4	Hunold	3 January 1990
5	Ernst	21 May 1991
6	Lorentz	7 February 1999
7	Mourgos	16 November 1999
8	Rajs	17 October 1995
9	Davies	29 January 1997
10	Matos	15 March 1998

19	Higgins	7 June 1994
20	Gietz	7 June 1994

ORACLE

```
SELECT  last_name,
 TO_CHAR(hire_date,
      'fmDdspth "of" Month YYYY fmHH:MI:SS AM')
 HIREDATE
FROM    employees;
```

Using the `TO_CHAR` Function with Numbers

```
TO_CHAR(number, 'format_model')
```

These are some of the format elements that you can use with the `TO_CHAR` function to display a number value as a character:

Element	Result
9	Represents a number
0	Forces a zero to be displayed
$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a comma as a thousands indicator

ORACLE

Using the `TO_CHAR` Function with Numbers

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

	SALARY
1	$6,000.00

ORACLE

Using the **TO_NUMBER** and **TO_DATE** Functions

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

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

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

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

- These functions have an fx modifier. This modifier
  specifies the exact match for the character argument and
  date format model of a TO_DATE function.

ORACLE

Using the `TO_CHAR` and `TO_DATE` Function with `RR` Date Format

To find employees hired before 1990, use the `RR` date format, which produces the same results whether the command is run in 1999 or now:

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM   employees
WHERE hire_date < TO_DATE('01-Jan-90','DD-Mon-RR');
```

	LAST_NAME	TO_CHAR(HIRE_DATE,'DD-MON-YYYY')
1	King	17-Jun-1987
2	Kochhar	21-Sep-1989
3	Whalen	17-Sep-1987

ORACLE

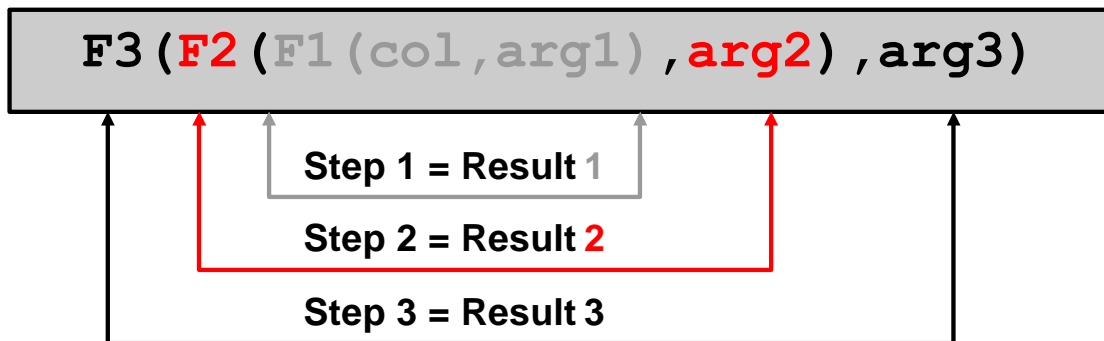Oracle University and BUSINESS SUPPORT LTDA use only

Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- **Nesting functions**
- General functions:
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
- Conditional expressions:
  - CASE
  - DECODE

ORACLE

Nesting Functions

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

$$F3(F2(F1(col,arg1),arg2),arg3)$$

Step 1 = Result 1

Step 2 = Result 2

Step 3 = Result 3

ORACLE

Nesting Functions

```
SELECT last_name,
  UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))
FROM    employees
WHERE   department_id = 60;
```

	LAST_NAME	UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US'))
1	Hunold	HUNOLD_US
2	Ernst	ERNST_US
3	Lorentz	LORENTZ_US

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Oracle Database 11*g*: SQL Fundamentals I 4 - 25

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions
- Nesting functions
- **General functions:**
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - `DECODE`

ORACLE

General Functions

The following functions work with any data type and pertain to using nulls:

- `NVL (expr1, expr2)`
- `NVL2 (expr1, expr2, expr3)`
- `NULLIF (expr1, expr2)`
- `COALESCE (expr1, expr2, ..., exprn)`

General Functions

These functions work with any data type and pertain to the use of null values in the expression list.

Function	Description
NVL	Converts a null value to an actual value
NVL2	If `expr1` is not null, `NVL2` returns `expr2`. If `expr1` is null, `NVL2` returns `expr3`. The argument `expr1` can have any data type.
NULLIF	Compares two expressions and returns null if they are equal; returns the first expression if they are not equal
COALESCE	Returns the first non-null expression in the expression list

Note: For more information about the hundreds of functions available, see the section on *Functions* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1).*

NVL Function

Converts a null value to an actual 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')`

ORACLE

Using the NVL Function

```
SELECT last_name, salary, NVL(commission_pct, 0),
    (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

(1)

(2)

	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
1	King	24000	0	288000
2	Kochhar	17000	0	204000
3	De Haan	17000	0	204000
4	Hunold	9000	0	108000
5	Ernst	6000	0	72000
6	Lorentz	4200	0	50400
7	Mourgos	5800	0	69600
8	Rajs	3500	0	42000
9	Davies	3100	0	37200
10	Matos	2600	0	31200
11	Vargas	2500	0	30000
12	Zlotkey	10500	0.2	151200

...

(1)   (2)

ORACLE

Using the `NVL2` Function

```
SELECT last_name,  salary, commission_pct,           1
       NVL2(commission_pct,
            'SAL+COMM', 'SAL') income           2
FROM    employees WHERE department_id IN (50, 80);
```

	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	Taylor	8600	0.2	SAL+COMM

1   2

ORACLE

Using the NULLIF Function

```
SELECT first_name, LENGTH(first_name) "expr1",
       last_name,  LENGTH(last_name)  "expr2",
       NULLIF(LENGTH(first_name), LENGTH(last_name)) result
FROM   employees;
```

(1)   (2)   (3)

	FIRST_NAME		expr1		LAST_NAME		expr2		RESULT
1	Ellen		5	Abel		4		5
2	Curtis		6	Davies		6		(null)
3	Lex		3	De Haan		7		3
4	Bruce		5	Ernst		5		(null)
5	Pat		3	Fay		3		(null)
6	William		7	Gietz		5		7
7	Kimberely		9	Grant		5		9

...

| 19 | Jennifer | | 8 | Whalen | | 6 | | 8 |
| 20 | Eleni | | 5 | Zlotkey | | 7 | | 5 |

(1)   (2)   (3)

Oracle University and BUSINESS SUPPORT LTDA use only

Using the COALESCE Function

- The advantage of the COALESCE function over the NVL function is that the COALESCE function can take multiple alternate values.
- If the first expression is not null, the COALESCE function returns that expression; otherwise, it does a COALESCE of the remaining expressions.

ORACLE

Using the COALESCE Function

```
SELECT last_name, employee_id,
COALESCE(TO_CHAR(commission_pct),TO_CHAR(manager_id),
         'No commission and no manager')
FROM employees;
```

	LAST_NAME	EMPLOYEE_ID	COALESCE(TO_CHAR(COMI
1	King	100	No commission and no manager
2	Kochhar	101	100
3	De Haan	102	100
4	Hunold	103	102
5	Ernst	104	103
6	Lorentz	107	103
7	Mourgos	124	100
8	Rajs	141	124

12	Zlotkey	149	.2
13	Abel	174	.3
14	Taylor	176	.2
15	Grant	178	.15
16	Whalen	200	101

...

ORACLE

Oracle Database 11*g*: SQL Fundamentals I 4 - 33

Conditional Expressions

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

ORACLE

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
```

Using the CASE Expression

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

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG'  THEN  1.10*salary
                   WHEN 'ST_CLERK' THEN  1.15*salary
                   WHEN 'SA_REP'   THEN  1.20*salary
       ELSE        salary END       "REVISED_SALARY"
FROM   employees;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...				
5	Ernst	IT_PROG	6000	6600
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
9	Davies	ST_CLERK	3100	3565
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
...				

ORACLE

Using the CASE Expression

In the SQL statement in the slide, the value of JOB_ID is decoded. If JOB_ID is IT_PROG, the salary increase is 10%; if JOB_ID is ST_CLERK, the salary increase is 15%; if JOB_ID is SA_REP, the salary increase is 20%. For all other job roles, there is no increase in salary.

The same statement can be written with the DECODE function.

The following code is an example of the searched CASE expression. In a searched CASE expression, the search occurs from left to right until an occurrence of the listed condition is found, and then it returns the return expression. If no condition is found to be true, and if an ELSE clause exists, the return expression in the ELSE clause is returned; otherwise, a NULL is returned.

```
SELECT last_name,salary,
(CASE WHEN salary<5000 THEN 'Low'
      WHEN salary<10000 THEN 'Medium'
      WHEN salary<20000 THEN 'Good'
      ELSE 'Excellent'
END) qualified_salary
FROM employees;
```

DECODE Function

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

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

Using the DECODE Function

```
SELECT last_name, job_id, salary,
       DECODE(job_id, 'IT_PROG',  1.10*salary,
                      'ST_CLERK', 1.15*salary,
                      'SA_REP',   1.20*salary,
              salary)
       REVISED_SALARY
FROM   employees;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...				
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
...				

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Oracle Database 11*g*: SQL Fundamentals I 4 - 40

5

Reporting Aggregated Data Using the Group Functions

Lesson Agenda

- Group functions:
  - Types and syntax
  - Use `AVG`, `SUM`, `MIN`, `MAX`, `COUNT`
  - Use `DISTINCT` keyword within group functions
  - `NULL` values in a group function
- Grouping rows:
  - `GROUP BY` clause
  - `HAVING` clause
- Nesting group functions

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS
COMPUTER IS STRICTLY PROHIBITED

Oracle Database 11*g*: SQL Fundamentals I 5 - 3

What Are Group Functions?

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

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	90	24000
2	90	17000
3	90	17000
4	60	9000
5	60	6000
6	60	4200
7	50	5800
8	50	3500
9	50	3100
10	50	2600

...

18	20	6000
19	110	12000
20	110	8300

Maximum salary in EMPLOYEES table

MAX(SALARY)
24000

ORACLE

What Are Group Functions?

Unlike single-row functions, group functions operate on sets of rows to give one result per group. These sets may comprise the entire table or the table split into groups.

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE
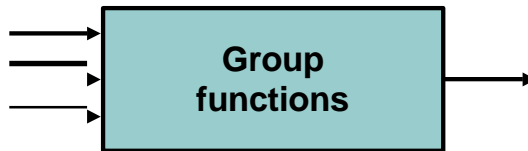
Group functions

ORACLE

Group Functions: Syntax

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

Using the AVG and SUM Functions

You can use AVG and SUM for numeric data.

```
SELECT  AVG(salary), MAX(salary),
        MIN(salary), SUM(salary)
FROM    employees
WHERE   job_id LIKE '%REP%';
```

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

ORACLE

Oracle Database 11g: SQL Fundamentals I 5 - 7

Using the MIN and MAX Functions

You can use MIN and MAX for numeric, character, and date data types.

```
SELECT  MIN(hire_date), MAX(hire_date)
FROM    employees;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	17-JUN-87	29-JAN-00

ORACLE

Using the COUNT Function

COUNT(*) returns the number of rows in a table:

1
```
SELECT  COUNT(*)
FROM    employees
WHERE   department_id = 50;
```

	COUNT(*)
1	5

COUNT(*expr*) returns the number of rows with non-null values for *expr*:

2
```
SELECT  COUNT(commission_pct)
FROM    employees
WHERE   department_id = 80;
```

	COUNT(COMMISSION_PCT)
1	3

ORACLE

Using the **DISTINCT** Keyword

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

- To display the number of distinct department values in the EMPLOYEES table:

```
SELECT COUNT(DISTINCT department_id)
FROM    employees;
```

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

ORACLE

Group Functions and Null Values

Group functions ignore null values in the column:

1
```
SELECT AVG(commission_pct)
FROM   employees;
```

	AVG(COMMISSION_PCT)
1	0.2125

The `NVL` function forces group functions to include null values:

2
```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

ORACLE

Lesson Agenda

- Group functions:
  - Types and syntax
  - Use `AVG, SUM, MIN, MAX, COUNT`
  - Use `DISTINCT` keyword within group functions
  - `NULL` values in a group function
- Grouping rows:
  - `GROUP BY` clause
  - `HAVING` clause
- Nesting group functions

Creating Groups of Data

EMPLOYEES

	DEPARTMENT_ID	SALARY	
1	10	4400	**4400**
2	20	13000	**9500**
3	20	6000	
4	50	5800	
5	50	2500	
6	50	2600	**3500**
7	50	3100	
8	50	3500	
9	60	4200	**6400**
10	60	6000	
11	60	9000	
12	80	11000	**10033**
13	80	10500	
14	80	8600	

. . .

19	110	12000	
20	(null)	7000	

Average salary in EMPLOYEES table for each department

	DEPARTMENT_ID	AVG(SALARY)
1	10	4400
2	20	9500
3	50	3500
4	60	6400
5	80	10033.333333333333...
6	90	19333.333333333333...
7	110	10150
8	(null)	7000

ORACLE

Creating Groups of Data:
GROUP BY Clause Syntax

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

You can divide rows in a table into smaller groups by using the GROUP BY clause.

Oracle University and BUSINESS SUPPORT LTDA use only

Using the GROUP BY Clause

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

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY department_id ;
```

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	90	19333.33333333333...
3	20	9500
4	110	10150
5	50	3500
6	80	10033.33333333333...
7	60	6400
8	10	4400

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Using the GROUP BY Clause

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

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id ;
```

	AVG(SALARY)
1	7000
2	19333.33333333333333333333...
3	9500
4	10150
5	3500
6	10033.33333333333333333333...
7	6400
8	4400

ORACLE

Grouping by More than One Column

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_MAN	5800
5	50	ST_CLERK	2500
6	50	ST_CLERK	2600
7	50	ST_CLERK	3100
8	50	ST_CLERK	3500
9	60	IT_PROG	4200
10	60	IT_PROG	6000
11	60	IT_PROG	9000
12	80	SA_REP	11000
13	80	SA_MAN	10500
14	80	SA_REP	8600

...

	DEPARTMENT_ID	JOB_ID	SALARY
19	110	AC_MGR	12000
20	(null)	SA_REP	7000

Add the salaries in the EMPLOYEES table for each job, grouped by department.

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	11700
5	50	ST_MAN	5800
6	60	IT_PROG	19200
7	80	SA_MAN	10500
8	80	SA_REP	19600
9	90	AD_PRES	24000
10	90	AD_VP	34000
11	110	AC_ACCOUNT	8300
12	110	AC_MGR	12000
13	(null)	SA_REP	7000

ORACLE

```
SELECT    department_id, job_id, sum(salary)
FROM      employees
GROUP BY department_id, job_id
ORDER BY job_id;
```

Oracle Database 11g: SQL Fundamentals I 5 - 17

Using the GROUP BY Clause on Multiple Columns

```
SELECT      department_id, job_id, SUM(salary)
FROM        employees
WHERE       department_id > 40
GROUP BY    department_id, job_id
ORDER BY    department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12000

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Check If valid?

```
SELECT department_id, COUNT(last_name)
FROM    employees;
```

ORA-00937: not a single-group group function
00937. 00000 - "not a single-group group function"

```
SELECT department_id, job_id, COUNT(last_name)
FROM    employees
GROUP BY department_id;
```
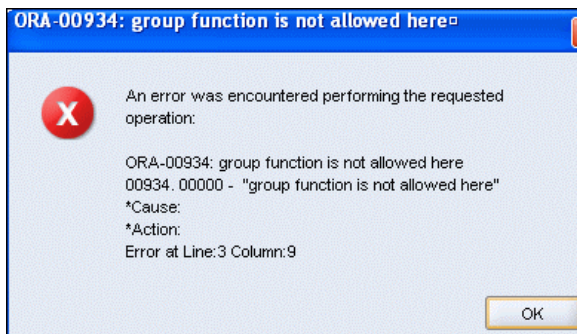
ORA-00979: not a GROUP BY expression
00979. 00000 - "not a GROUP BY expression"

**Either add `job_id` in the `GROUP BY` or
remove the `job_id` column from the
`SELECT` list.**

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

```
SELECT    department_id, AVG(salary)
FROM      employees
WHERE     AVG(salary) > 8000
GROUP BY department_id;
```

ORA-00934: group function is not allowed here

An error was encountered performing the requested operation:

ORA-00934: group function is not allowed here
00934. 00000 - "group function is not allowed here"
*Cause:
*Action:
Error at Line:3 Column:9

OK

ORACLE

Restricting Group Results

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	5800
5	50	2500
6	50	2600
7	50	3100
8	50	3500
9	60	4200
10	60	6000
11	60	9000
12	80	11000
13	80	10500
14	80	8600

18	110	8300
19	110	12000
20	(null)	7000

The maximum salary per department when it is greater than $10,000

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	80	11000
3	90	24000
4	110	12000

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Restricting Group Results
with the HAVING Clause

When you use the HAVING clause, the Oracle server restricts groups as follows:

1. Rows are grouped.

2. The group function is applied.

3. 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

```
SELECT     department_id, MAX(salary)
FROM       employees
GROUP BY department_id
HAVING     MAX(salary)>10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12000
4	80	11000

ORACLE

Using the `HAVING` Clause

```
SELECT    job_id, SUM(salary) PAYROLL
FROM      employees
WHERE     job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING    SUM(salary) > 13000
ORDER BY SUM(salary);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

ORACLE

Nesting Group Functions

Display the maximum average salary:

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

	MAX(AVG(SALARY))
1	19333.3333333333333333333333333333333333

ORACLE

Quiz

Identify the guidelines for group functions and the `GROUP BY` clause.

1. You cannot use a column alias in the `GROUP BY` clause.
2. The `GROUP BY` column must be in the `SELECT` clause.
3. By using a `WHERE` clause, you can exclude rows before dividing them into groups.
4. The `GROUP BY` clause groups rows and ensures order of the result set.
5. If you include a group function in a `SELECT` clause, you cannot select individual results as well.

ORACLE

Answer: 1, 3

6

Displaying Data
from Multiple Tables

Oracle University and BUSINESS SUPPORT LTDA use only

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
...			
18	202	Fay	20
19	205	Higgins	110
20	206	Gietz	110

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping
5	144	50	Shipping
...			
18	205	110	Accounting
19	206	110	Accounting

ORACLE

Types of Joins

Joins that are compliant with the SQL:1999 standard include the following:

- Natural joins:
  - NATURAL JOIN clause
  - USING clause
  - ON clause
- OUTER joins:
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN
- Cross joins

Joining Tables Using SQL:1999 Syntax

Use a join to query data from more than one table:

```
SELECT    table1.column, table2.column
FROM      table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)]|
[CROSS JOIN table2];
```

ORACLE

Qualifying Ambiguous
Column Names

- Use table prefixes to qualify column names that are in multiple tables.

- Use table prefixes to improve performance.

- Instead of full table name prefixes, use table aliases.

- Table alias gives a table a shorter name:
  - Keeps SQL code smaller, uses less memory

- Use column aliases to distinguish columns that have identical names, but reside in different tables.

ORACLE

Creating Natural Joins

- The `NATURAL JOIN` clause is based on all columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.

ORACLE

Retrieving Records with Natural Joins

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford

ORACLE

Creating Joins with the `USING` Clause

- If several columns have the same names but the data types do not match, use the `USING` clause to specify the columns for the equijoin.
- Use the `USING` clause to match only one column when more than one column matches.
- The `NATURAL JOIN` and `USING` clauses are mutually exclusive.

ORACLE

Joining Column Names

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
100	90
101	90
102	90
103	60
104	60
107	60
124	50
141	50
142	50
143	50
144	50
149	80
174	80
176	80

…

Foreign key

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

Primary key

ORACLE

Retrieving Records with the USING Clause

```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	124	Mourgos	1500	50
5	144	Vargas	1500	50
6	143	Matos	1500	50
7	142	Davies	1500	50
8	141	Rajs	1500	50
9	107	Lorentz	1400	60
10	104	Ernst	1400	60

...

19	205	Higgins	1700	110

ORACLE

Using Table Aliases with the `USING` Clause

- Do not qualify a column that is used in the `USING` clause.
- If the same column is used elsewhere in the SQL statement, do not alias it.

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d
USING (location_id)
WHERE d.location_id = 1400;
```


ORA-25154: column part of USING clause cannot have qualifier

An error was encountered performing the requested operation:

ORA-25154: column part of USING clause cannot have qualifier
25154. 00000 - "column part of USING clause cannot have qualifier"
*Cause: Columns that are used for a named-join (either a NATURAL join
          or a join with a USING clause) cannot have an explicit qualifier.
*Action: Remove the qualifier.
Error at Line:4 Column:6

ORACLE

Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.
- The ON clause makes code easy to understand.

ORACLE

Retrieving Records with the ON Clause

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400

...

ORACLE

Creating Three-Way Joins with the ON Clause

```
SELECT employee_id, city, department_name
FROM    employees e
JOIN    departments d
ON      d.department_id = e.department_id
JOIN    locations l
ON      d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping

...

ORACLE

Applying Additional Conditions to a Join

Use the AND clause or the WHERE clause to apply additional conditions:

```
SELECT  e.employee_id, e.last_name, e.department_id,
        d.department_id, d.location_id
FROM    employees e JOIN departments d
ON      (e.department_id = d.department_id)
AND     e.manager_id = 149 ;
```

Or

```
SELECT  e.employee_id, e.last_name, e.department_id,
        d.department_id, d.location_id
FROM    employees e JOIN departments d
ON      (e.department_id = d.department_id)
WHERE    e.manager_id = 149 ;
```

ORACLE

Joining a Table to Itself

EMPLOYEES (WORKER)

	EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	100	King	(null)
2	101	Kochhar	100
3	102	De Haan	100
4	103	Hunold	102
5	104	Ernst	103
6	107	Lorentz	103
7	124	Mourgos	100
8	141	Rajs	124
9	142	Davies	124
10	143	Matos	124

...

EMPLOYEES (MANAGER)

	EMPLOYEE_ID	LAST_NAME
	100	King
	101	Kochhar
	102	De Haan
	103	Hunold
	104	Ernst
	107	Lorentz
	124	Mourgos
	141	Rajs
	142	Davies
	143	Matos

...

**MANAGER_ID in the WORKER table is equal to
EMPLOYEE_ID in the MANAGER table.**

ORACLE

Self-Joins Using the ON Clause

```
SELECT worker.last_name emp, manager.last_name mgr
FROM    employees worker JOIN employees manager
ON      (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King
9	Hartstein	King
10	De Haan	King

…

ORACLE

Nonequijoins

EMPLOYEES

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hunold	9000
5	Ernst	6000
6	Lorentz	4200
7	Mourgos	5800
8	Rajs	3500
9	Davies	3100
10	Matos	2600

...

19	Higgins	12000
20	Gietz	8300

JOB_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

`JOB_GRADES` table defines the `LOWEST_SAL` and `HIGHEST_SAL` range of values for each `GRADE_LEVEL`. Hence, the `GRADE_LEVEL` column can be used to assign grades to each employee.

ORACLE

Retrieving Records with Nonequijoins

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

...

ORACLE

Oracle University and BUSINESS SUPPORT LTDA use only

Oracle Database 11*g*: SQL Fundamentals I 6 - 24

Returning Records with No Direct Match
Using OUTER Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

Equijoin with EMPLOYEES

	DEPARTMENT_ID	LAST_NAME
1	90	King
2	90	Kochhar
3	90	De Haan
4	60	Hunold
5	60	Ernst
6	60	Lorentz
7	50	Mourgos
8	50	Rajs
9	50	Davies
10	50	Matos

...

	DEPARTMENT_ID	LAST_NAME
18	110	Higgins
19	110	Gietz

**There are no employees
in department 190.**

**Employee "Grant" has
not been assigned a
department ID.**

ORACLE

INNER Versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an INNER join.

- A join between two tables that returns the results of the INNER join as well as the unmatched rows from the left (or right) table is called a left (or right) OUTER join.

- A join between two tables that returns the results of an INNER join as well as the results of a left and right join is a full OUTER join.

ORACLE

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM    employees e LEFT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

ORACLE

RIGHT OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM    employees e RIGHT OUTER JOIN departments d
ON      (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Mourgos	50	Shipping

. . .

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	(null)	190	Contracting

ORACLE

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON   (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Hunold	60	IT

...

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
15	Grant	(null)	(null)
16	Whalen	10	Administration
17	Hartstein	20	Marketing
18	Fay	20	Marketing
19	Higgins	110	Accounting
20	Gietz	110	Accounting
21	(null)	190	Contracting

ORACLE

Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition.

ORACLE

Generating a Cartesian Product

EMPLOYEES (20 rows)

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
4	103	Hunold	60

. . .

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
19	205	Higgins	110
20	206	Gietz	110

DEPARTMENTS (8 rows)

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

Cartesian product:

20 x 8 = 160 rows

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	100	90	1700
2	101	90	1700
3	102	90	1700
4	103	60	1700

. . .

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
159	205	110	1700
160	206	110	1700

ORACLE

Creating Cross Joins

- The CROSS JOIN clause produces the cross-product of two tables.
- This is also called a Cartesian product between the two tables.

```
SELECT last_name, department_name
FROM    employees
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration

...

159	Whalen	Contracting
160	Zlotkey	Contracting

ORACLE

Quiz

The SQL:1999 standard join syntax supports the following
types of joins. Which of these join types does Oracle join syntax
support?

1. Equijoins
2. Nonequijoins
3. Left OUTER join
4. Right OUTER join
5. Full OUTER join
6. Self joins
7. Natural joins
8. Cartesian products

7

Using Subqueries to Solve Queries

ORACLE®

Oracle University and BUSINESS SUPPORT LTDA use only

Using a Subquery to Solve a Problem

Who has a salary greater than Abel's?

Main query:

Which employees have salaries greater than Abel's salary?

Subquery:

What is Abel's salary?

Oracle University and BUSINESS SUPPORT LTDA use only
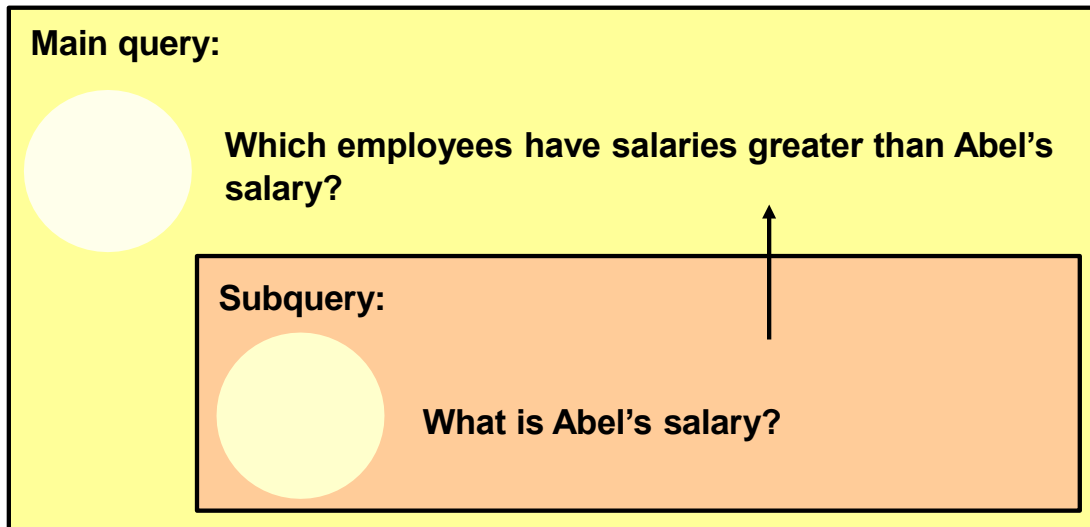
Subquery Syntax

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

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

ORACLE

Using a Subquery

```
SELECT last_name, salary
FROM    employees
WHERE   salary >   11000
                (SELECT salary
                 FROM    employees
                 WHERE   last_name = 'Abel');
```

	LAST_NAME		SALARY
1	King		24000
2	Kochhar		17000
3	De Haan		17000
4	Hartstein		13000
5	Higgins		12000

ORACLE

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition for readability (However, the subquery can appear on either side of the comparison operator.).
- Use single-row operators with single-row subqueries and multiple-row operators with multiple-row subqueries.

ORACLE

Types of Subqueries

- Single-row subquery

- Multiple-row subquery

ORACLE

Single-Row Subqueries
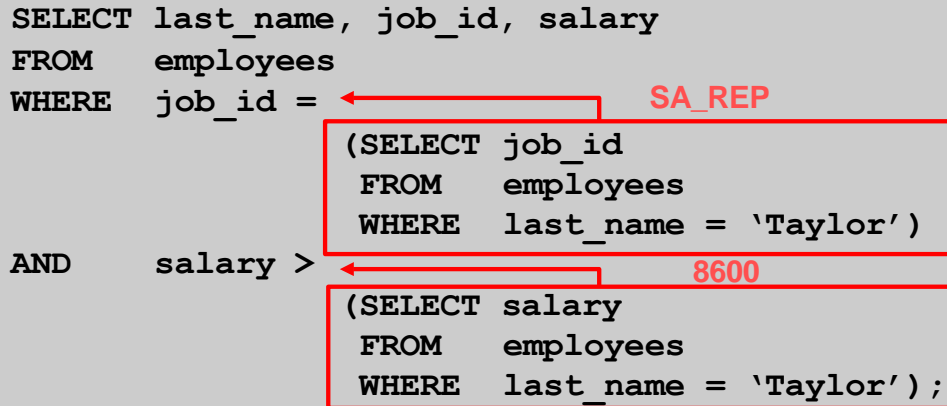
- Return only one row
- Use single-row comparison operators

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

ORACLE

Executing Single-Row Subqueries

```
SELECT last_name, job_id, salary
FROM    employees
WHERE   job_id =                          SA_REP
                     (SELECT job_id
                      FROM    employees
                      WHERE   last_name = 'Taylor')
AND     salary >                          8600
                     (SELECT salary
                      FROM    employees
                      WHERE   last_name = 'Taylor');
```

	LAST_NAME		JOB_ID		SALARY
1	Abel		SA_REP		11000

ORACLE

Using Group Functions in a Subquery

```
SELECT last_name, job_id, salary
FROM    employees
WHERE   salary =                    2500
                (SELECT MIN(salary)
                 FROM    employees);
```

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

The **HAVING** Clause with Subqueries

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

```
SELECT    department_id, MIN(salary)
FROM      employees
GROUP BY department_id                    2500
HAVING    MIN(salary) >
                        (SELECT MIN(salary)
                         FROM    employees
                         WHERE   department_id = 50);
```

	DEPARTMENT_ID	MIN(SALARY)
1	(null)	7000
2	90	17000
3	20	6000
...		
7	10	4400

ORACLE

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle Database 11*g*: SQL Fundamentals I 7 - 13

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
FROM    employees
WHERE   salary =
                (SELECT   MIN(salary)
                 FROM        employees
                 GROUP BY department_id);
```

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

An error was encountered performing the requested operation:

ORA-01427: single-row subquery returns more than one row
01427. 00000 - "single-row subquery returns more than one row"
*Cause:
*Action:
Error at Line:1

ORACLE

Oracle Database 11g: SQL Fundamentals I 7 - 14

No Rows Returned by the Inner Query

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Haas');

0 rows selected
```

**Subquery returns no rows because there is no
employee named "Haas."**

Multiple-Row Subqueries

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

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. Compares a value to each value in a list or returned by a query. Evaluates to FALSE if the query returns no rows.
ALL	Must be preceded by =, !=, >, <, <=, >=. Compares a value to every value in a list or returned by a query. Evaluates to TRUE if the query returns no rows.

ORACLE

Using the `ANY` Operator in Multiple-Row Subqueries

```
SELECT  employee_id, last_name, job_id, salary
FROM    employees           9000, 6000, 4200
WHERE   salary < ANY
                (SELECT salary
                 FROM    employees
                 WHERE   job_id = 'IT_PROG')
AND     job_id <> 'IT_PROG';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

ORACLE

Using the `ALL` Operator
in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM    employees        9000, 6000, 4200
WHERE   salary < ALL

              (SELECT salary
               FROM    employees
               WHERE   job_id = 'IT_PROG')
AND     job_id <> 'IT_PROG';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141 Rajs	ST_CLERK	3500
2	142 Davies	ST_CLERK	3100
3	143 Matos	ST_CLERK	2600
4	144 Vargas	ST_CLERK	2500

Null Values in a Subquery

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                              (SELECT mgr.manager_id
                               FROM   employees mgr);
```

```
0 rows selected
```

Quiz

Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search value(s) in the second query.

1. True
2. False

Oracle University and BUSINESS SUPPORT LTDA use only

8

Using the Set Operators

Set Operators

UNION/UNION ALL

INTERSECT

MINUS
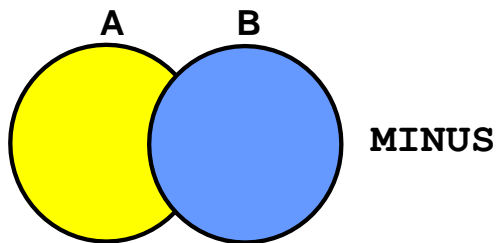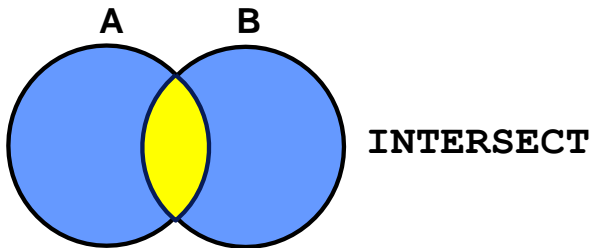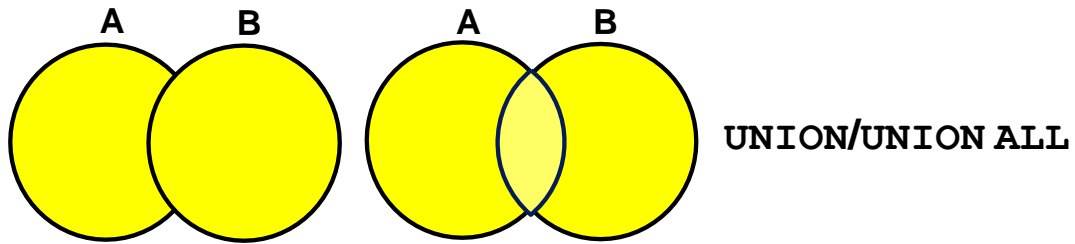
ORACLE

Set Operator Guidelines

- The expressions in the SELECT lists must match in number.
- The data type of each column in the second query must match the data type of its corresponding column in the first query.
- Parentheses can be used to alter the sequence of execution.
- ORDER BY clause can appear only at the very end of the statement.

ORACLE

The Oracle Server and Set Operators

- Duplicate rows are automatically eliminated except in UNION ALL.

- Column names from the first query appear in the result.

- The output is sorted in ascending order by default except in UNION ALL.

ORACLE

Lesson Agenda

- Set Operators: Types and guidelines
- **Tables used in this lesson**
- `UNION` and `UNION ALL` operator
- `INTERSECT` operator
- `MINUS` operator
- Matching the `SELECT` statements
- Using the `ORDER BY` clause in set operations

Oracle University and BUSINESS SUPPORT LTDA use only

Tables Used in This Lesson

The tables used in this lesson are:

- `EMPLOYEES`: Provides details regarding all current employees
- `JOB_HISTORY`: Records the details of the start date and end date of the former job, and the job identification number and department when an employee switches jobs

UNION Operator

A   B
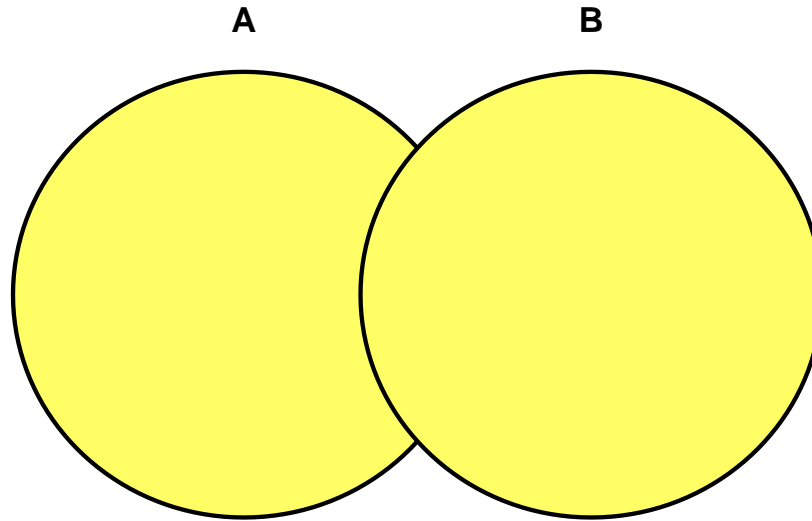
The `UNION` operator returns rows from both queries after eliminating duplications.

Using the UNION Operator

Display the current and previous job details of all employees.
Display each employee only once.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

	EMPLOYEE_ID	JOB_ID
1	100	AD_PRES
2	101	AC_ACCOUNT

22	200	AC_ACCOUNT
23	200	AD_ASST
24	201	MK_MAN

...

ORACLE
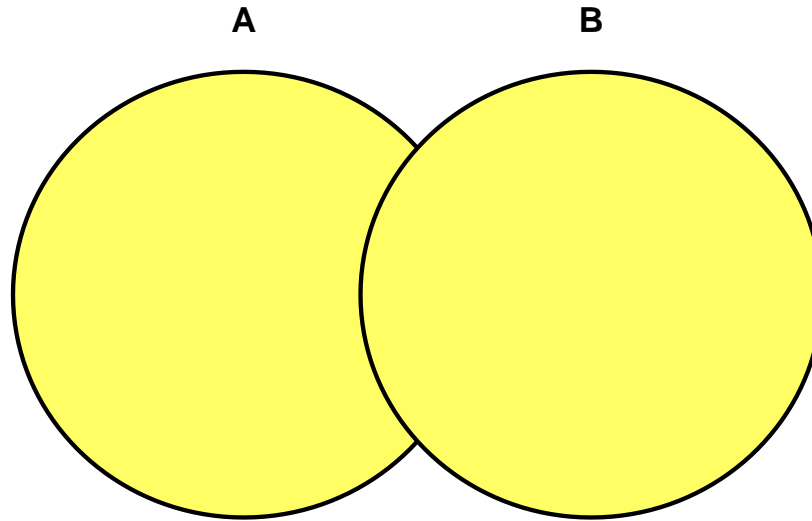
UNION ALL Operator

The UNION ALL operator returns rows from both queries, including all duplications.

ORACLE

Using the UNION ALL Operator

Display the current and previous departments of all employees.

```
SELECT employee_id, job_id, department_id
FROM    employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM    job_history
ORDER BY  employee_id;
```

	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100	AD_PRES	90
16	144	ST_CLERK	50
17	149	SA_MAN	80
18	174	SA_REP	80
19	176	SA_REP	80
20	176	SA_MAN	80
21	176	SA_REP	80
22	178	SA_REP	(null)
60	206	AC_ACCOUNT	110

ORACLE

INTERSECT Operator
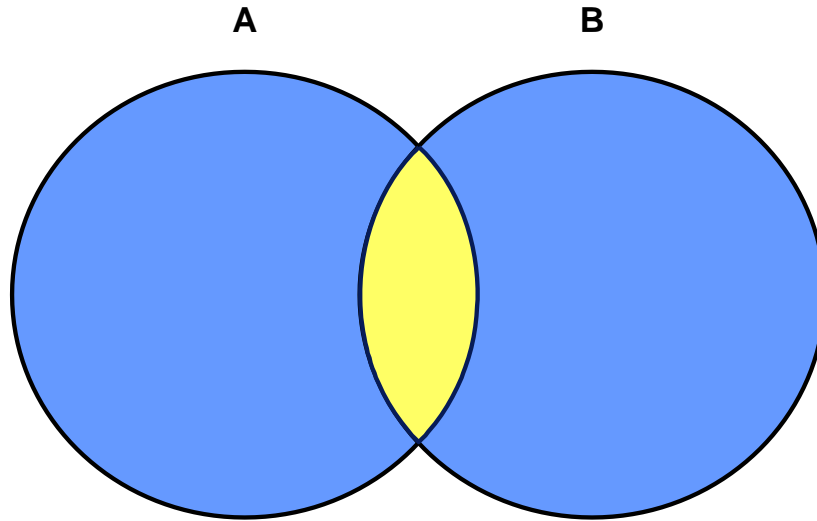
The INTERSECT operator returns rows that are common to both queries.

ORACLE

Using the INTERSECT Operator

Display the employee IDs and job IDs of those employees who currently have a job title that is the same as their previous one (that is, they changed jobs but have now gone back to doing the same job they did previously).
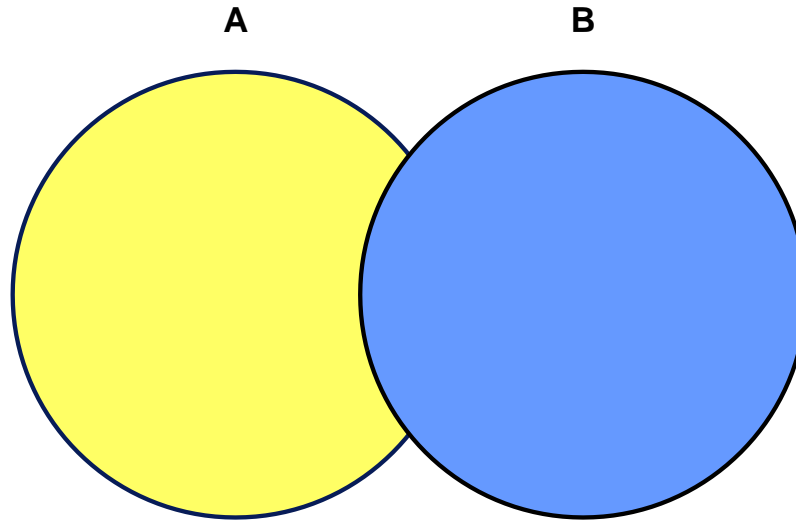
```
SELECT employee_id, job_id
FROM    employees
INTERSECT
SELECT employee_id, job_id
FROM    job_history;
```

	EMPLOYEE_ID	JOB_ID
1	176	SA_REP
2	200	AD_ASST

ORACLE

MINUS Operator

A   **B**

The `MINUS` operator returns all the distinct rows selected by the first query, but not present in the second query result set.

ORACLE

Using the `MINUS` Operator

Display the employee IDs of those employees who have not changed their jobs even once.

```
SELECT  employee_id
FROM    employees
MINUS
SELECT  employee_id
FROM    job_history;
```

	EMPLOYEE_ID
1	100
2	103
3	104
4	107
5	124

14	205
15	206

ORACLE

Matching the `SELECT` Statements

- Using the `UNION` operator, display the location ID, department name, and the state where it is located.
- You must match the data type (using the `TO_CHAR` function or any other conversion functions) when columns do not exist in one or the other table.

```
SELECT location_id, department_name "Department",
   TO_CHAR(NULL) "Warehouse location"
FROM departments
UNION
SELECT location_id, TO_CHAR(NULL) "Department",
   state_province
FROM locations;
```

ORACLE

Matching the `SELECT` Statement: Example

Using the `UNION` operator, display the employee ID, job ID, and salary of all employees.

```
SELECT employee_id, job_id,salary
FROM   employees
UNION
SELECT employee_id, job_id,0
FROM   job_history;
```

	EMPLOYEE_ID	JOB_ID	SALARY
1	100	AD_PRES	24000
2	101	AC_ACCOUNT	0
3	101	AC_MGR	0
4	101	AD_VP	17000
5	102	AD_VP	17000

29	205	AC_MGR	12000
30	206	AC_ACCOUNT	8300

ORACLE

Using the ORDER BY Clause in Set Operations

- The ORDER BY clause can appear only once at the end of the compound query.
- Component queries cannot have individual ORDER BY clauses.
- ORDER BY clause recognizes only the columns of the first SELECT query.
- By default, the first column of the first SELECT query is used to sort the output in an ascending order.

ORACLE

Quiz

Identify the set operator guidelines.

1. The expressions in the `SELECT` lists must match in number.

2. Parentheses may not be used to alter the sequence of execution.

3. The data type of each column in the second query must match the data type of its corresponding column in the first query.

4. The `ORDER BY` clause can be used only once in a compound query, unless a `UNION ALL` operator is used.

ORACLE