Practice 1: Overview

This practice covers the following topics:

- Selecting all data from different tables
- Describing the structure of tables
- Performing arithmetic calculations and specifying column names
- Using iSQL*Plus

Restricting and Sorting Data

Objectives

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

- Limit the rows that are retrieved by a query
- Sort the rows that are retrieved by a query
- Use ampersand substitution in iSQL*Plus to restrict and sort output at run time

Limiting Rows Using a Selection

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
104	Ernst	IT_PROG	60
107	Lorentz	IT_PROG	60
124	Mourgos	ST_MAN	50

20 rows selected.

"retrieve all employees in department 90"

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

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.

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
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Character Strings and Dates

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

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

Comparison Conditions

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

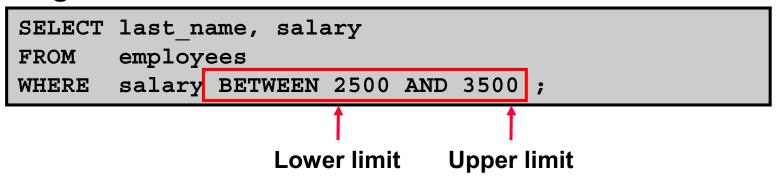
Using Comparison Conditions

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

LAST_NAME	SALARY
Matos	2600
Vargas	2500

Using the BETWEEN Condition

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



LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

Using the IN Condition

Use the IN membership condition to test for values in a list:

```
SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 101, 201);
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

8 rows selected.

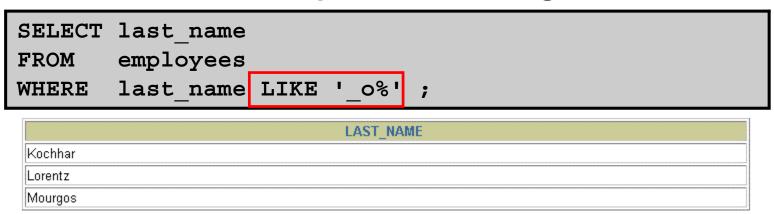
Using the LIKE Condition

- Use the LIKE condition 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%';
```

Using the LIKE Condition

You can combine pattern-matching characters:



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

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
King	

Logical Conditions

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

Using the AND Operator

AND requires both 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
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

Using the OR Operator

OR requires either 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
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

8 rows selected.

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
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

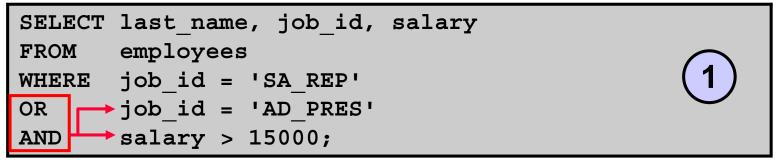
¹⁰ rows selected.

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.

Rules of Precedence



LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Abel	SA_REP	11000
Taylor	SA_REP	8600
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;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000

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;
```

JOB_ID	DEPARTMENT_ID	HIRE_DATE
AD_PRES	90	17-JUN-87
AD_ASST	10	17-SEP-87
AD_VP	90	21-SEP-89
IT_PROG	60	03-JAN-90
IT_PROG	60	21-MAY-91
	AD_PRES AD_ASST AD_VP IT_PROG	AD_PRES 90 AD_ASST 10 AD_VP 90 IT_PROG 60

- - -

20 rows selected.

Sorting

Sorting in descending order:

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

Sorting by column alias:

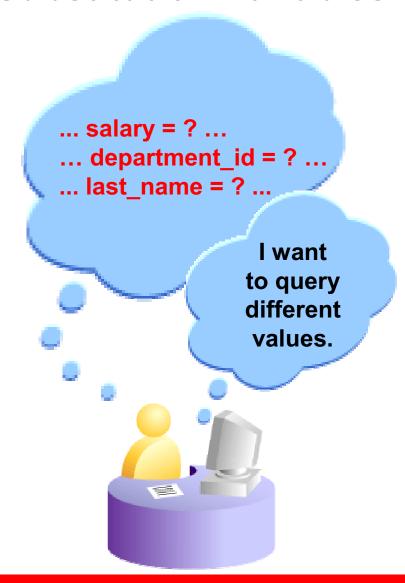
```
SELECT employee_id, last_name, salary*12 annsal FROM employees
ORDER BY annsal;
```

Sorting by multiple columns:

```
SELECT last_name, department_id, salary
FROM employees

ORDER BY department_id, salary DESC;
```

Substitution Variables

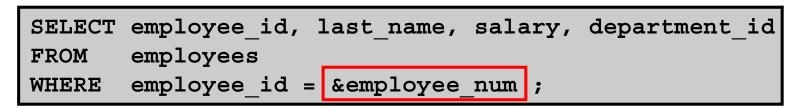


Substitution Variables

- Use iSQL*Plus 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

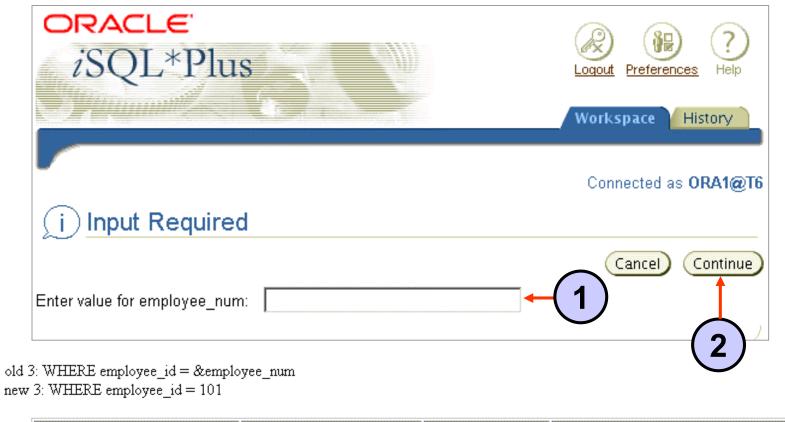
Using the & Substitution Variable

Use a variable prefixed with an ampersand (&) to prompt the user for a value:



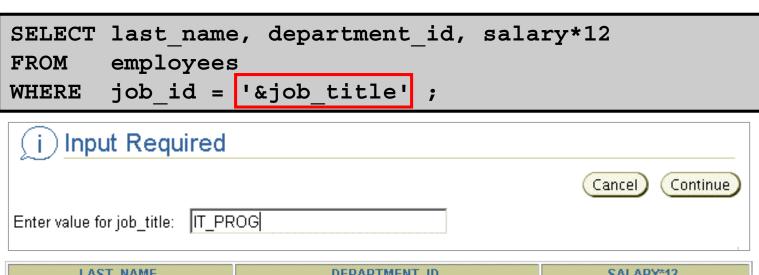
	Connected as ORA1@T6
i Input Required	
	Cancel Continue
Enter value for employee_num:	

Using the & Substitution Variable



Character and Date Values with Substitution Variables

Use single quotation marks for date and character values:



LAST_NAME	DEPARTMENT_ID	SALARY*12
Hunold	60	108000
Ernst	60	72000
Lorentz	60	50400

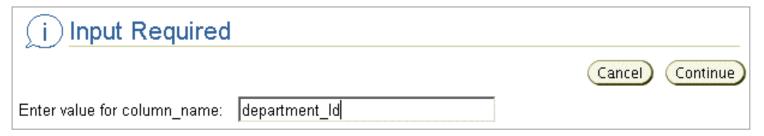
Specifying Column Names, Expressions, and Text

SELECT employee_id, last_name, job_id,& FROM employees WHERE &condition ORDER BY ℴ_column;	column_name
i Input Required Enter value for column_name: salary	Cancel Continue
Enter value for condition: salary > 15000	Cancel Continue
Enter value for order_column: last_name	Cancel Continue

Using the && Substitution Variable

Use the double ampersand (&&) if you want to reuse the variable value without prompting the user each time:

SELECT	employee_id,	last_name,	job_id,	$\&\&column_name$
FROM	employees			
ORDER BY	&column_name	;		



EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
200	Whalen	AD_ASST	10
201	Hartstein	MK_MAN	20

20 rows selected.

Using the iSQL*Plus DEFINE Command

- Use the iSQL*Plus DEFINE command to create and assign a value to a variable.
- Use the *i*SQL*Plus 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 VERIFY Command

Use the VERIFY command to toggle the display of the substitution variable, both before and after *i*SQL*Plus replaces substitution variables with values:

```
SET VERIFY ON
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num;
```

```
"employee_num" 200
```

```
old 3: WHERE employee_id = &employee_num
new 3: WHERE employee_id = 200
```

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]];
```

 Use ampersand substitution in iSQL*Plus to restrict and sort output at run time

Practice 2: Overview

This practice covers the following topics:

- Selecting data and changing the order of the rows that are displayed
- Restricting rows by using the WHERE clause
- Sorting rows by using the ORDER BY clause
- Using substitution variables to add flexibility to your SQL SELECT statements