LAB # 6: INTRODUCTION TO SQL

(RETRIEVING, RESTRICTING, Sorting AND OPERATORS)

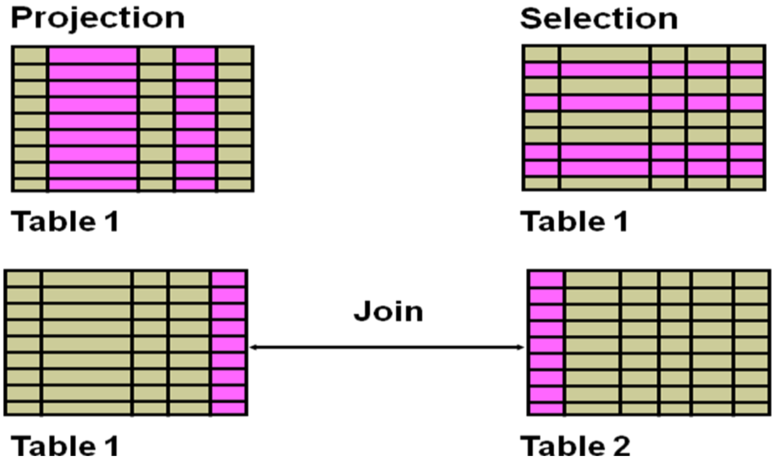
Objective:

1. Database Login and sample schema
2. Introduction to SQL (DML, DDL, DCL)
3. SQL sample commands and interactions
4. Learning and practice

Capabilities of SQL SELECT Statements

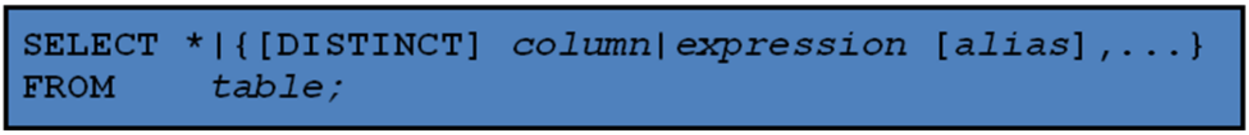
A SELECT statement retrieves information from the database. With a SELECT statement, you can use the following capabilities:

- **Projection:** Select the columns in a table that are returned by a query. Select as few or as many of the columns as required.
- **Selection:** Select the rows in a table that are returned by a query. Various criteria can be used to restrict the rows that are retrieved.
- **Joining:** Bring together data that is stored in different tables by specifying the link between them. SQL joins are covered in more detail in the lesson titled —Displaying Data from Multiple Tables.



Basic SELECT Statement

Syntax:



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

In its simplest form, a SELECT statement must include the following:

- A SELECT clause, which specifies the columns to be displayed
- A FROM clause, which identifies the table containing the columns that are listed in the SELECT clause

In the syntax:

SELECT is a list of one or more columns

***** selects all columns

DISTINCT suppresses duplicates

column|expression selects the named column or the expression ***alias*** gives the selected columns different headings **FROM** ***table*** specifies

the table containing the columns

Note: Throughout this course, the words *keyword*, *clause*, and *statement* are used as follows:

- A *keyword* refers to an individual SQL element. For example, SELECT and FROM are keywords.
- A *clause* is a part of a SQL statement. For example, SELECT employee_id, last_name, and so on is a clause.
- A *statement* is a combination of two or more clauses. For example, SELECT * FROM employees is a SQL statement.

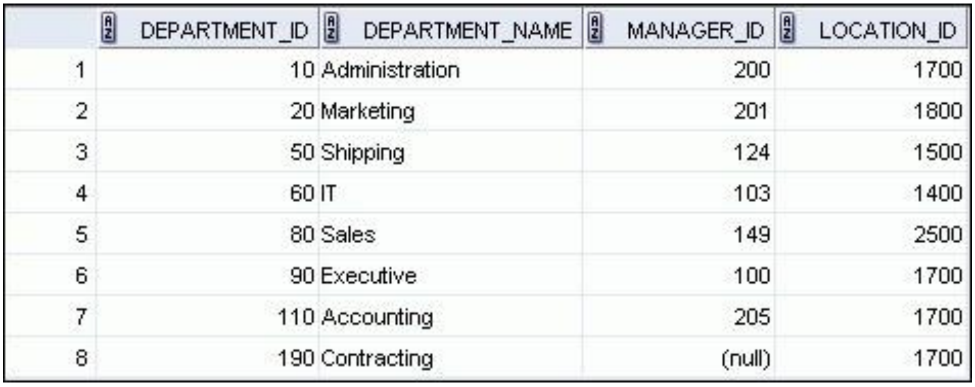
Selecting All Columns

You can display all columns of data in a table by following the SELECT keyword with an asterisk (*). The department table contains four columns: DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, and LOCATION_ID. The table contains eight rows, one for each department.

Example:

SELECT * FROM departments

Output:



	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

You can also display all columns in the table by listing all the columns after the SELECT keyword. For example, the following SQL statement (like the example in the slide) displays all columns and all rows of the DEPARTMENTS table:

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

Selecting Specific Columns

You can use the SELECT statement to display specific columns of the table by specifying the column names, separated by commas. Above example displays all the department numbers and location numbers from the DEPARTMENTS table.

In the SELECT clause, specify the columns that you want in the order in which you want them to appear in the output. For example, to display location before department number (from left to right), you use the following statement:

Example:

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

Writing SQL Statements

By using the following simple rules and guidelines, you can construct valid statements that are both easy to read and edit:

- SQL statements are not case-sensitive (unless indicated).
- SQL statements can be entered on one or many lines.
- Keywords cannot be split across lines or abbreviated.
- Clauses are usually placed on separate lines for readability and ease of editing.
- Indents should be used to make code more readable.
- Keywords typically are entered in uppercase; all other words, such as table names and columns names are entered in lowercase.

Arithmetic Expressions

You may need to modify the way in which data is displayed, or you may want to perform calculations, or look at what-if scenarios. All these are possible using arithmetic expressions. An arithmetic expression can contain column names, constant numeric values, and the arithmetic operators.

Arithmetic Operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic Operators

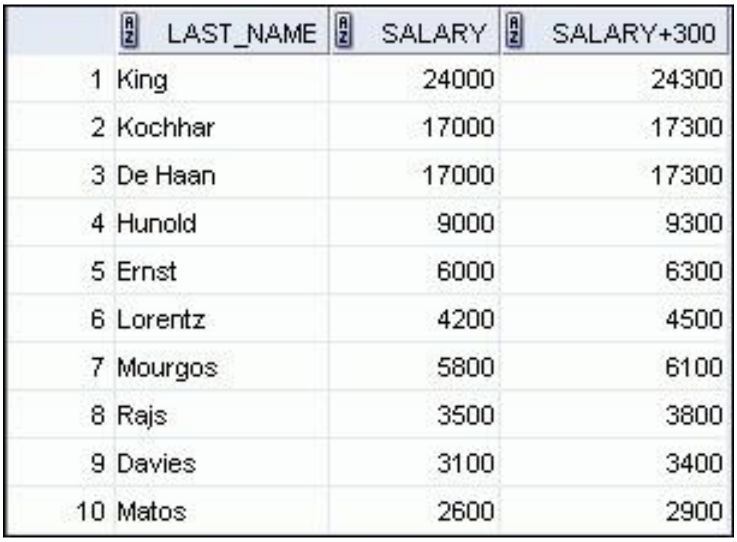
Example:

```
SELECT last_name, salary, salary + 300  FROM   employees;
```

This example uses the addition operator to calculate a salary increase of $300 for all employees. The slide also displays a SALARY+300 column in the output.

Note that the resultant calculated column, SALARY+300, is not a new column in the EMPLOYEES table; it is for display only. By default, the name of a new column comes from the calculation that generated it—in this case, salary+300.

Note: The MySQL server ignores blank spaces before and after the arithmetic operator.



	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900

Output

Operator Precedence

If an arithmetic expression contains more than one operator, multiplication and division are evaluated first. If operators in an expression are of the same priority, then evaluation is done from left to right.

You can use parentheses to force the expression that is enclosed by the parentheses to be evaluated first.

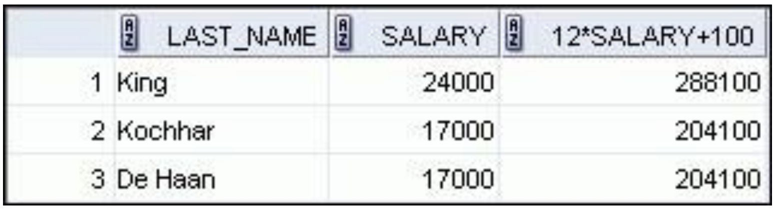
Rules of Precedence:

- Multiplication and division occur before addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to override the default precedence or to clarify the statement.

Example 1:

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

Output:



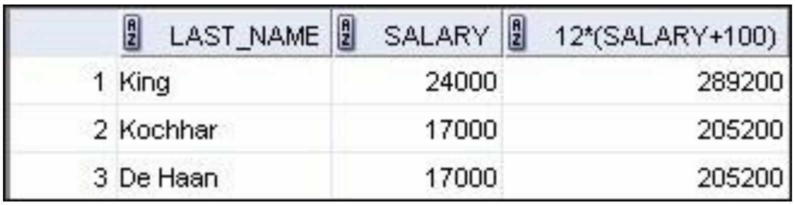
	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100

Example 2:

SELECT last_name, salary, 12*(salary+100)

FROM   employees;

Output:



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

The first example in the slide displays the last name, salary, and annual compensation of employees. It calculates the annual compensation by multiplying the monthly salary with 12, plus a one-time bonus of $100. Note that multiplication is performed before addition.

Note: Use parentheses to reinforce the standard order of precedence and to improve clarity. For example, the expression in the slide can be written as (12*salary)+100 with no change in the result.

Using Parentheses

You can override the rules of precedence by using parentheses to specify the desired order in which the operators are to be executed.

The second example in the slide displays the last name, salary, and annual compensation of employees. It calculates the annual compensation as follows: adding a monthly bonus of $100 to the monthly salary, and then multiplying that subtotal with 12. Because of the parentheses, addition takes priority over multiplication.

Defining a Null Value

If a row lacks a data value for a particular column, that value is said to be *null* or to contain a null.

Null is a value that is unavailable, unassigned, unknown, or inapplicable. Null is not the same as zero or a blank space. Zero is a number and blank space is a character.

Columns of any data type can contain nulls. However, some constraints (NOT NULL and PRIMARY KEY) prevent nulls from being used in the column.

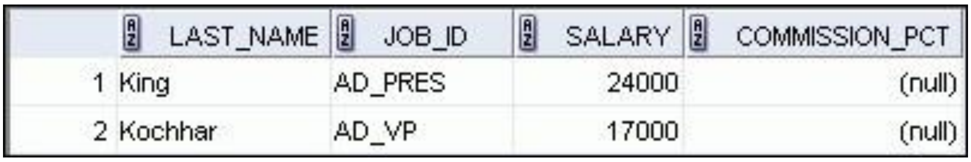
In the COMMISSION_PCT column in the EMPLOYEES table, notice that only a sales manager or sales representative can earn a commission. Other employees are not entitled to earn commissions. A null represents that fact.

Example:

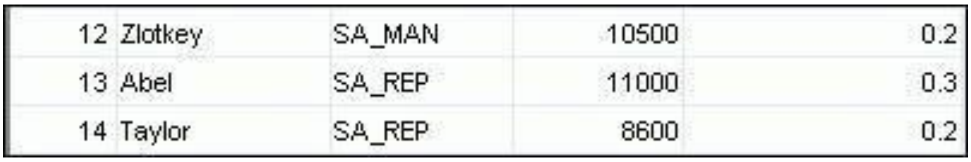
SELECT last_name, job_id, salary, commission_pct

FROM   employees;

Output:



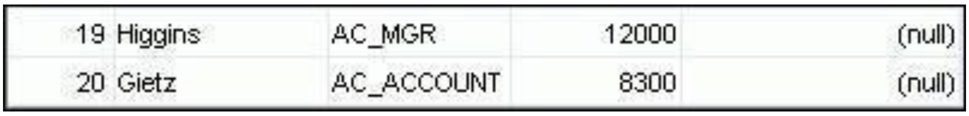
	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)



12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2

▪▪▪

▪▪▪



19	Higgins	AC_MGR	12000	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

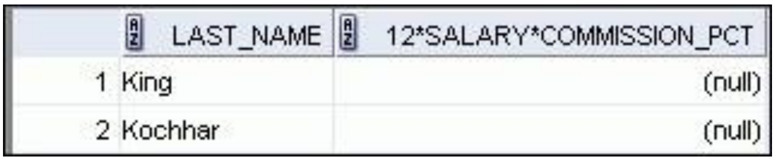
Null Values in Arithmetic Expressions

If any column value in an arithmetic expression is null, the result is null. For example, if you attempt to perform division by zero, you get an error. However, if you divide a number by null, the result is a null or unknown.

Example:

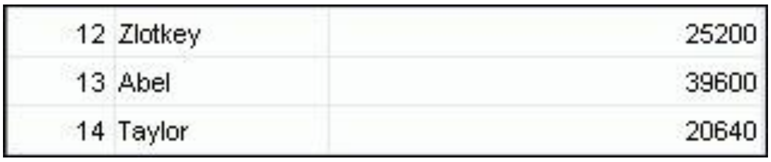
```
SELECT last_name, 12*salary*commission_pct
FROM   employees;
```

Output:



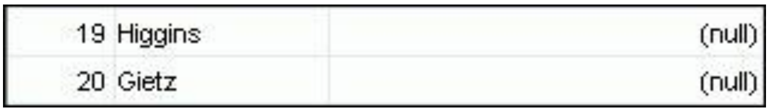
	LAST_NAME	12*SALARY*COMMISSION_PCT
1	King	(null)
2	Kochhar	(null)

▪▪▪



12	Zlotkey	25200
13	Abel	39600
14	Taylor	20640

▪▪▪



19	Higgins	(null)
20	Gietz	(null)

In the example, employee King does not get any commission. Because the COMMISSION_PCT column in the arithmetic expression is null, the result is null.

Column Alias

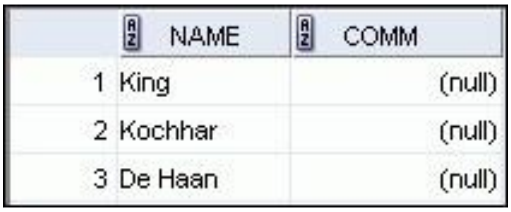
When displaying the result of a query, SQL Developer normally uses the name of the selected column as the column heading. This heading may not be descriptive and, therefore, may be difficult to understand. You can change a column heading by using a column alias.

Specify the alias after the column in the SELECT list using blank space as a separator. By default, alias headings appear in uppercase. If the alias contains spaces or special characters (such as # or $), or if it is case-sensitive, enclose the alias in double quotation marks (— ‖).

Example 1:

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

Output:



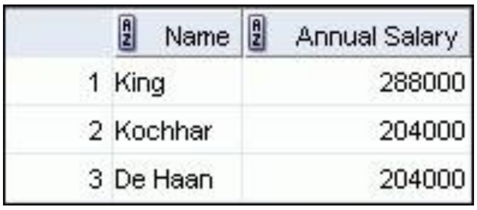
	NAME	COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)

■ ■ ■

Example 2:

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

Output:



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

■ ■ ■

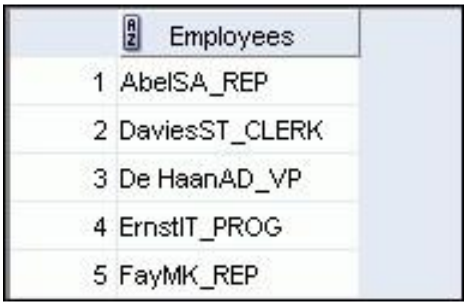
Concatenation Operator

You can link columns to other columns, arithmetic expressions, or constant values to create a character expression by using the concatenation operator (||). Columns on either side of the operator are combined to make a single output column.

Example:

```
SELECT concat(last_name,job_id) AS
"Employees" FROM employees;
```

Output:



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

In the example, LAST_NAME and JOB_ID are concatenated, and given the alias Employees. Note that the last name of the employee and the job code are combined to make a single output column.

The AS keyword before the alias name makes the SELECT clause easier to read.

Null Values with the Concatenation Operator

If you concatenate a null value with a character string, the result is a character string. LAST_NAME || NULL results in LAST_NAME.

Note: You can also concatenate date expressions with other expressions or columns.

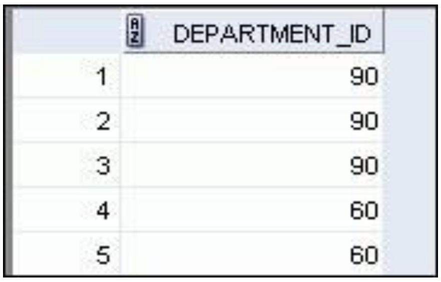
Duplicate Rows

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

Example:

```
SELECT department_id
FROM   employees;
```

Output:



	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60

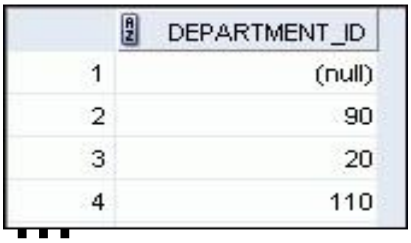
This example displays all the department numbers from the EMPLOYEES table. Note that the department numbers are repeated.

To eliminate duplicate rows in the result, include the DISTINCT keyword in the SELECT clause immediately after the SELECT keyword.

Example:

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

Output:



	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110

...

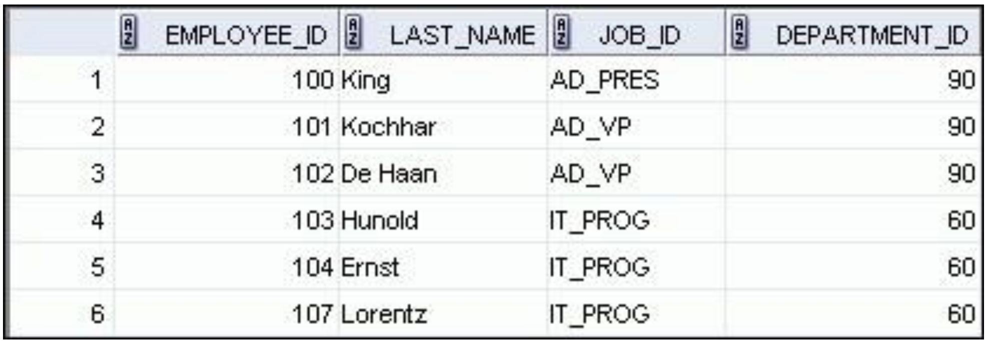
In the above example, the EMPLOYEES table actually contains 20 rows, but there are only seven unique department numbers in the table.

You can specify multiple columns after the DISTINCT qualifier. The DISTINCT qualifier affects all the selected columns, and the result is every distinct combination of the columns.

```
SELECT  DISTINCT department_id, job_id
FROM    employees;
```

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

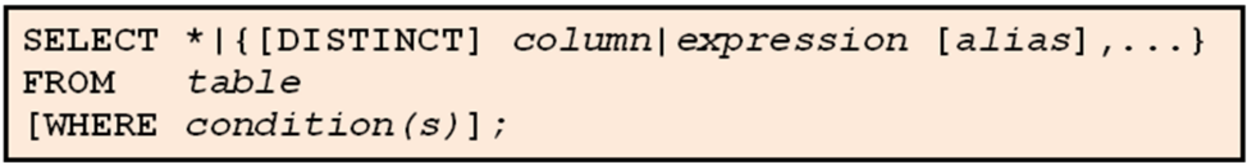
...

Assume that you want to display all the employees in department 90. The rows with a value of 90 in the DEPARTMENT_ID column are the only ones that are returned. This method of restriction is the basis of the WHERE clause in SQL.

Limiting the Rows That Are Selected

You can restrict the rows that are returned from the query by using the WHERE clause. A WHERE clause contains a condition that must be met and it directly follows the FROM clause. If the condition is true, the row meeting the condition is returned.

Syntax:



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

In the syntax:

WHERE restricts the query to rows that meet a condition

condition is composed of column names, expressions, constants, and a comparison operator. A condition specifies a combination of one or more expressions and logical (Boolean) operators, and returns a value of TRUE, FALSE, or UNKNOWN.

The WHERE clause can compare values in columns, literal, arithmetic expressions, or functions. It consists of three elements:

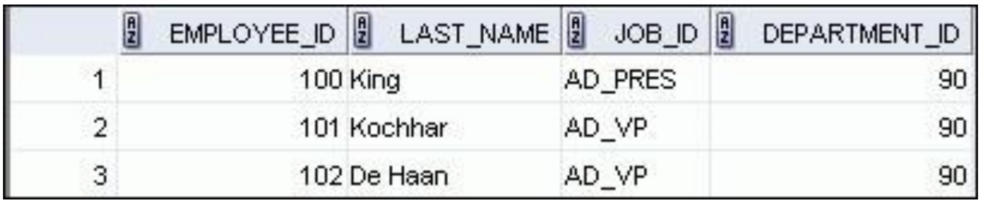
- Column name
- Comparison condition
- Column name, constant, or list of values

Using the WHERE Clause

Example:

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

Output:



	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

In this example, the SELECT statement retrieves the employee ID, last name, job ID, and department number of all employees who are in department 90.

Character Strings and Dates

Character strings and dates in the WHERE clause must be enclosed with single quotation marks (''). Number constants, however, should not be enclosed with single quotation marks.

All character searches are case-sensitive. In the following example, no rows are returned because the EMPLOYEES table stores all the last names in mixed case:

Example:

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

MySQL retrieves and displays DATE values in ' YYYY-MM-DD ' format. The supported range is '1000-01-01' to '9999-12-31' . The DATETIME type is used for values that contain both date and time parts. MySQL retrieves and displays DATETIME values in ' YYYY-MM-DD hh:mm:ss ' format.

Comparison Operators

Comparison operators are used in conditions that compare one expression to another value or expression. They are used in the WHERE clause in the following format:

Syntax

... WHERE *expr operator value*

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

Example

... WHERE hire_date = '01-JAN-95'

... WHERE salary >= 6000

... WHERE last_name = 'Smith'

An alias cannot be used in the WHERE clause.

Note: The symbols != and ^= can also represent the *not equal to* condition.

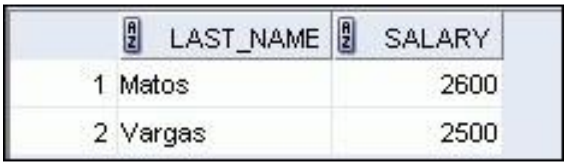
Using Comparison Operators

Example:

```
SELECT last_name, salary
FROM   employees

WHERE  salary <= 3000 ;
```

Output:



	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

In this example, the SELECT statement retrieves the last name and salary from the EMPLOYEES table for any employee whose salary is less than or equal to $3,000. Note that there is an explicit value supplied to the WHERE clause. The explicit value of 3000 is compared to the salary value in the SALARY column of the EMPLOYEES table.

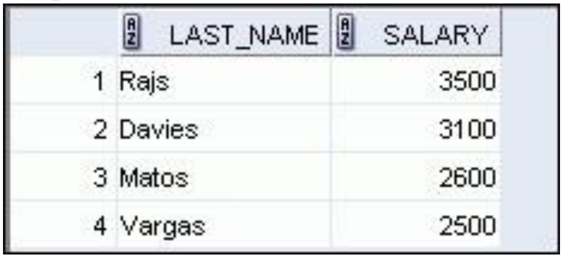
Range Conditions Using the BETWEEN Operator

You can display rows based on a range of values using the BETWEEN operator. The range that you specify contains a lower limit and an upper limit.

Example:

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

Output:



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

The SELECT statement in the above example returns rows from the EMPLOYEES table for any employee whose salary is between $2,500 and $3,500.

Values that are specified with the BETWEEN operator are inclusive. However, you must specify the lower limit first.

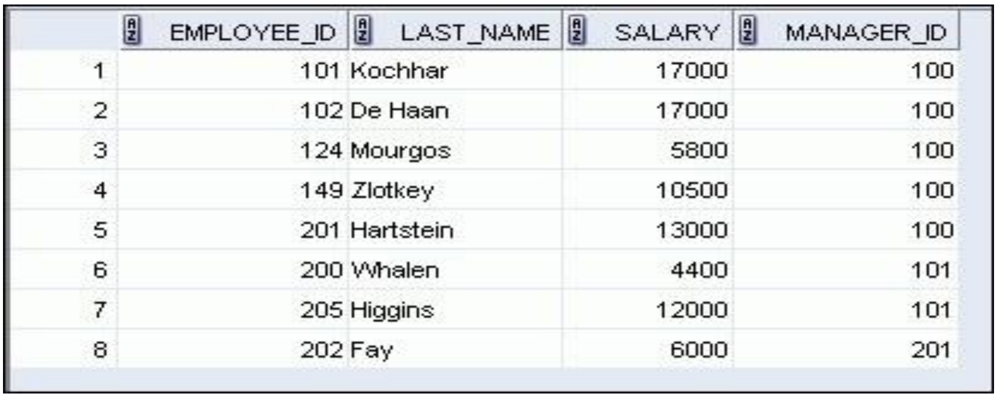
Membership Condition Using the IN Operator

To test for values in a specified set of values, use the IN operator. The condition defined using the IN operator is also known as the *membership condition*.

Example:

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

Output:



	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

The example displays employee numbers, last names, salaries, and managers‘ employee numbers for all the employees whose manager‘s employee number is 100, 101, or 201.

The IN operator can be used with any data type. The following example returns a row from the EMPLOYEES table, for any employee whose last name is included in the list of names in the WHERE clause:

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

If characters or dates are used in the list, they must be enclosed with single quotation marks ('').

Note: The IN operator is internally evaluated by the Mysql server as a set of OR conditions, such as a=value1 or a=value2 or a=value3. Therefore, using the IN operator has no performance benefits and is used only for logical simplicity.

Pattern Matching Using the LIKE Operator

You may not always know the exact value to search for. You can select rows that match a character pattern by using the LIKE operator. The character pattern–matching operation is referred to as a *wildcard* search.

Search conditions can contain either literal characters or numbers:

- % denotes zero or many characters.
- _ denotes one character.

Example:

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

The SELECT statement in the example returns the first name from the EMPLOYEES table for any employee whose first name begins with the letter —S.‖ Note the uppercase —S.‖ Consequently, names beginning with a lowercase —s‖ are not returned.

The LIKE operator can be used as a shortcut for some BETWEEN comparisons. The following example displays the last names and hire dates of all employees who joined between January, 1995 and December, 1995:

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date LIKE '%95';
```

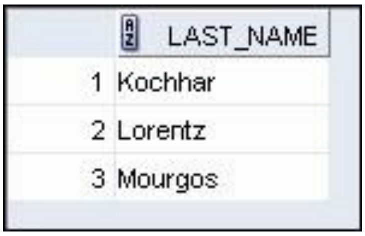
Combining Wildcard Characters

The % and _ symbols can be used in any combination with literal characters.

Example:

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

Output:



```
  LAST_NAME
1 Kochhar
2 Lorentz
3 Mourgos
```

The above example displays the names of all employees whose last names have the letter —o‖ as the second character.

ESCAPE Identifier

When you need to have an exact match for the actual % and _ characters, use the ESCAPE identifier. This option specifies what the escape character is. If you want to search for strings that contain SA_, you can use the following SQL statement:

```
SELECT employee_id, last_name, job_id
FROM   employees WHERE  job_id LIKE '%SA\_%' ESCAPE '\';
```

The ESCAPE identifier identifies the backslash (\) as the escape character. In the SQL statement, the escape character precedes the underscore (_). This causes the Mysql server to interpret the underscore literally.

Using the NULL Conditions

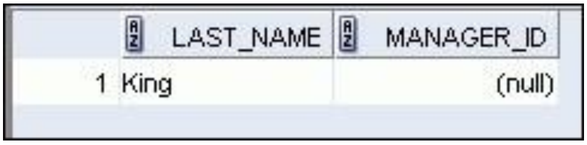
The NULL conditions include the IS NULL condition and the IS NOT NULL condition.

The IS NULL condition tests for nulls. A null value means that the value is unavailable, unassigned, unknown, or inapplicable. Therefore, you cannot test with =, because a null cannot be equal or unequal to any value.

Example:

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

Output:



	LAST_NAME	MANAGER_ID
1	King	(null)

The above example retrieves the last names and managers of all employees who do not have a manager.

Here is another example: To display the last name, job ID, and commission for all employees who are *not* entitled to receive a commission, use the following SQL statement:

Example:

```
SELECT last_name, job_id, commission_pct
FROM   employees
WHERE  commission_pct IS NULL;
```

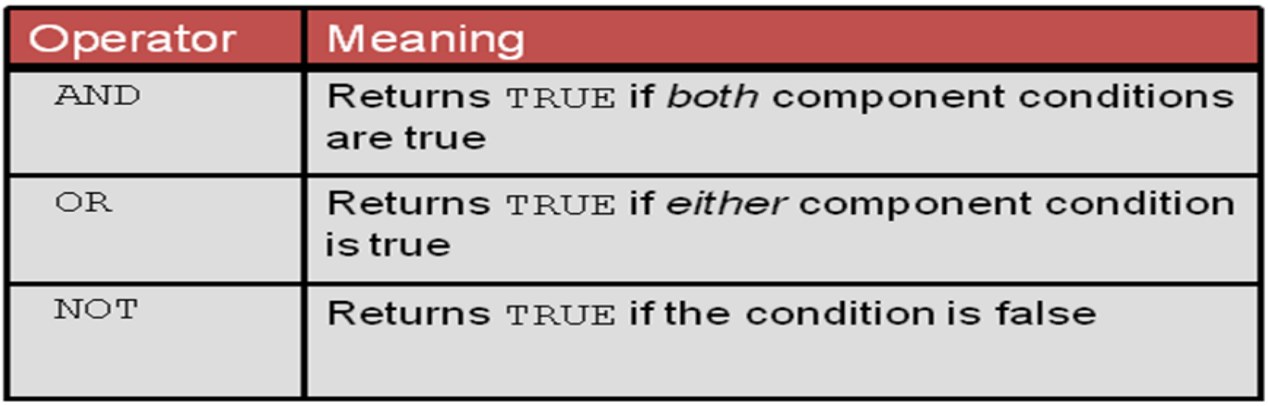
Defining Conditions Using the Logical Operators

A logical condition combines the result of two component conditions to produce a single result based on those conditions or it inverts the result of a single condition. A row is returned only if the overall result of the condition is true.

Three logical operators are available in SQL:

- AND
- OR
- NOT

All the examples so far have specified only one condition in the WHERE clause. You can use several conditions in a single WHERE clause using the AND and OR 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

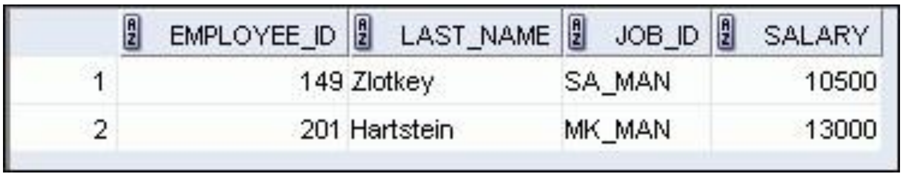
AND operator requires both the component conditions to be true.

Example:

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000

AND   job_id LIKE '%MAN%';
```

Output:



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

In the example, both the component conditions must be true for any record to be selected. Therefore, only those employees who have a job title that contains the string _MAN‘ *and* earn $10,000 or more are selected.

All character searches are case-sensitive, that is no rows are returned if _MAN‘ is not uppercase. Further, character strings must be enclosed with quotation marks.

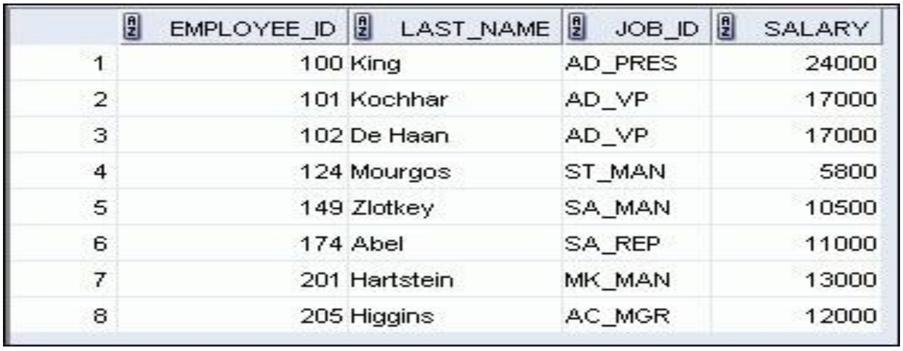
Using the OR Operator

OR operator requires either component condition to be true.

Example:

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

Output:



	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

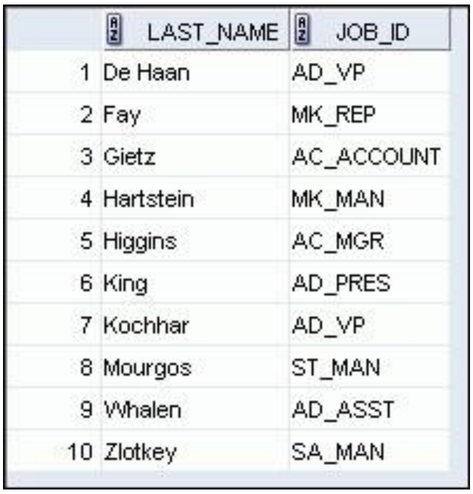
In the example, either component condition can be true for any record to be selected. Therefore, any employee who has a job ID that contains the string _MAN‘ *or* earns $10,000 or more is selected.

Using the NOT Operator

Example:

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

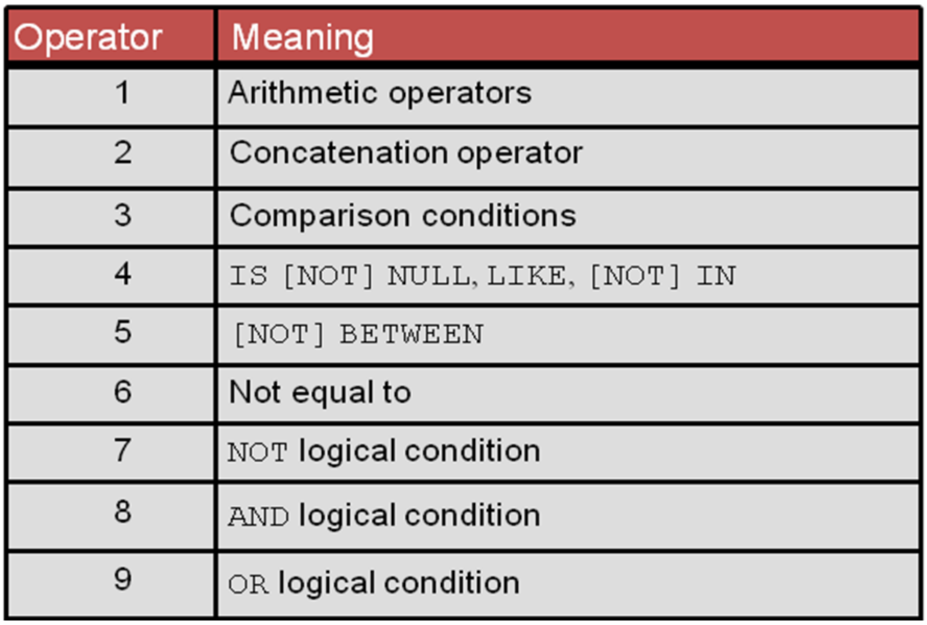
Output:



	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

Rules of Precedence

The rules of precedence determine the order in which expressions are evaluated and calculated. The following table lists the default order of precedence. However, you can override the default order by using parentheses around the expressions that you want to calculate first.

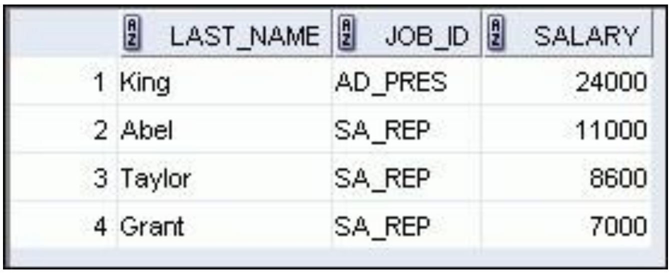


Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical condition
8	AND logical condition
9	OR logical condition

Example 1:

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

Output:



	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

In this example, there are two conditions:

- The first condition is that the job ID is AD_PRES *and* the salary is greater than $15,000.
- The second condition is that the job ID is SA_REP.

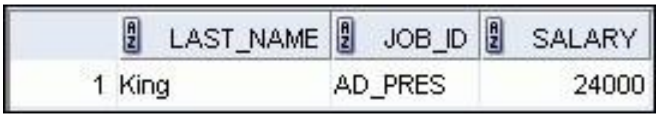
Therefore, the SELECT statement reads as follows:

—Select the row if an employee is a president *and* earns more than $15,000, *or* if the employee is a sales representative.‖

Example 2:

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

Output:



	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000

In this example, there are two conditions:

- The first condition is that the job ID is AD_PRES *or* SA_REP.
- The second condition is that the salary is greater than $15,000.

Therefore, the SELECT statement reads as follows:

—Select the row if an employee is a president *or* a sales representative, *and* if the employee earns more than $15,000.‖

Sorting rows using the ORDER BY clause

The order of rows that are returned in a query result is undefined. The ORDER BY clause can be used to sort the rows. However, if you use the ORDER BY clause, it must be the last clause of the SQL statement. Further, you can specify an expression, an alias, or a column position as the sort condition.

Syntax

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

In the syntax:

ORDER BY specifies the order in which the retrieved rows are displayed

ASC orders the rows in ascending order (this is the default order)

DESC orders the rows in descending order

If the ORDER BY clause is not used, the sort order is undefined, and the Mysql server may not fetch rows in the same order for the same query twice. Use the ORDER BY clause to display the rows in a specific order.

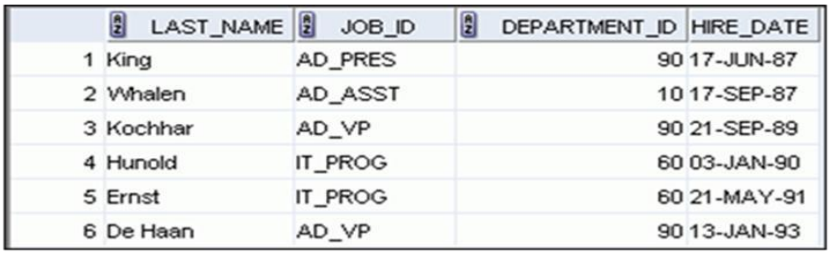
Note: Use the keywords NULLS FIRST or NULLS LAST to specify whether returned rows containing null values should appear first or last in the ordering sequence.

Example:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees

ORDER BY hire_date ;
```

Output:



	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

...

Sorting in descending order:

Example:

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

To reverse the order in which the rows are displayed, specify the DESC keyword after the column name in the ORDER BY clause. The slide example sorts the result by the most recently hired employee.

Sorting by column alias:

Example:

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

You can also use a column alias in the ORDER BY clause. The above example sorts the data by annual salary.

Sorting by using the column’s numeric position:

Example:

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

You can sort query results by specifying the numeric position of the column in the SELECT clause. The example sorts the result by the department_id as this column is at the third position in the SELECT clause.

Sorting by multiple columns:

Example:

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

You can sort query results by more than one column. The sort limit is the number of columns in the given table. In the ORDER BY clause, specify the columns and separate the column names using commas. If you want to reverse the order of a column, specify DESC after its name