



Course Code: CL-2005	Course: Database Systems Lab
Instructor:	Yasir Arfat

## Contents:

1. Database
2. SQL
3. Basic SQL Concepts

## Database

A database is a systematic collection of data. They support electronic storage and manipulation of data. Databases make data management easy.

### Example #1

An online telephone directory uses a database to store data of people, phone numbers, and other contact details. Your electricity service provider uses a database to manage billing, client-related issues, handle fault data, etc.

### Example #2

Facebook needs to store, manipulate, and present data related to members, their friends, member activities, messages, advertisements, and a lot more. We can provide a countless number of examples for the usage of databases.

## SQL

**SQL** is the standard language for dealing with Relational Databases. SQL can be used to insert, search, update, and delete database records. SQL can do lots of other operations, including optimizing and maintenance of databases. SQL stands for Structured Query language, pronounced as "S-Q-L" or sometimes as "See-Quel"... Relational databases like MySQL Database, Oracle, MS SQL Server, Sybase, etc. use ANSI SQL.

## Basic SQL Concepts

### Data Types

- String :

Data Type	Description
-----------	-------------

<b>CHAR(size)</b>	A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters (0 to 255). Default is 1.
<b>VARCHAR(size)</b>	A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters (0 to 65535).
<b>BINARY(size)</b>	Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1.
<b>VARBINARY(size)</b>	Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.
<b>TINYBLOB</b>	For BLOBs (Binary Large Objects). Max length: 255 bytes.
<b>TINYTEXT</b>	Holds a string with a maximum length of 255 characters.
<b>TEXT(size)</b>	Holds a string with a maximum length of 65,535 bytes.
<b>BLOB(size)</b>	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data.
<b>MEDIUMTEXT</b>	Holds a string with a maximum length of 16,777,215 characters.
<b>MEDIUMBLOB</b>	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data.
<b>LONGTEXT</b>	Holds a string with a maximum length of 4,294,967,295 characters.
<b>LOBLOB</b>	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data.
<b>ENUM(val1, val2, ...)</b>	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them.
<b>SET(val1, val2, ...)</b>	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list.

- **Numeric:**

Data Type	Description
<b>BIT(size)</b>	A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.
<b>TINYINT(size)</b>	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255).
<b>BOOL</b>	Zero is considered as false, nonzero values are considered as true.
<b>BOOLEAN</b>	Equal to BOOL.
<b>SMALLINT(size)</b>	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255).
<b>MEDIUMINT(size)</b>	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255).
<b>INT(size)</b>	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255).
<b>INTEGER(size)</b>	Equal to INT(size).
<b>BIGINT(size)</b>	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255).
<b>FLOAT(size, d)</b>	A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions.
<b>FLOAT(p)</b>	A floating point number. MySQL uses the value of p to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT. If p is from 25 to 53, the data type becomes DOUBLE.
<b>DOUBLE(size, d)</b>	A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter.
<b>DOUBLE PRECISION(size, d)</b>	Equal to DOUBLE(size, d).
<b>DECIMAL(size, d)</b>	An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for d is 0.
<b>DEC(size, d)</b>	Equal to DECIMAL(size, d).

- **Date and time:**

Data Type	Description
<b>DATE</b>	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'.
<b>DATETIME(fsp)</b>	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition provides automatic initialization and updating to the current date and time.
<b>TIMESTAMP(fsp)</b>	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition.
<b>TIME(fsp)</b>	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'.
<b>YEAR</b>	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support the year in two-digit format.

## I. Writing SQL Statements

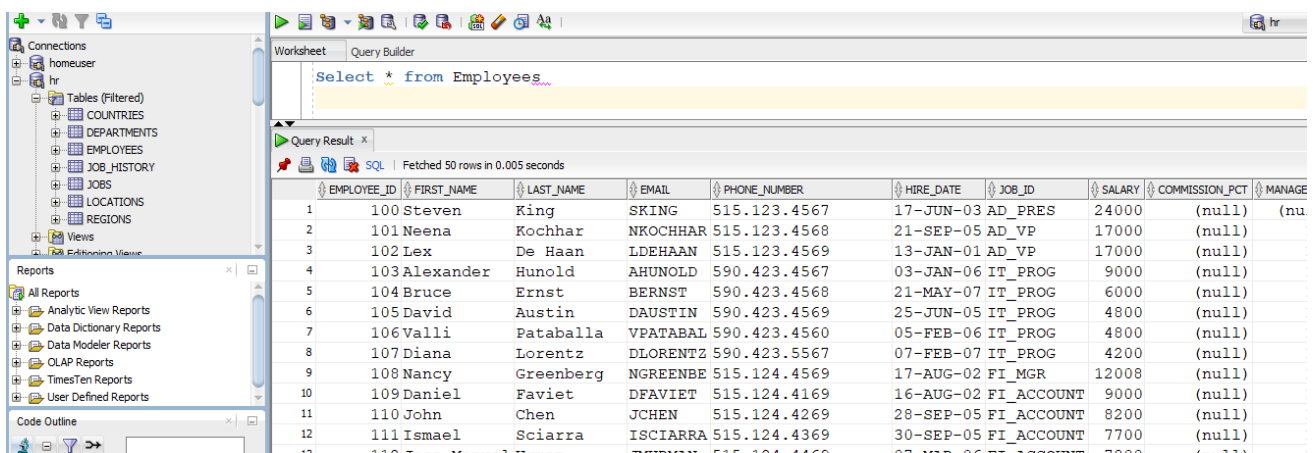
By using the following simple rules and guidelines, you can construct valid statements that are both easy to read and edit:

- 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.

## II. Basic SQL Queries

**Note: Connect the HR Database in SqlDeveloper**

**Select \* from EMPLOYEES**



The screenshot shows the SQL Developer interface. The 'Query Builder' window displays the SQL statement 'Select \* from Employees'. The 'Query Result' window shows the fetched data for the first 12 rows of the EMPLOYEES table. The table has columns: EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, EMAIL, PHONE\_NUMBER, HIRE\_DATE, JOB\_ID, SALARY, COMMISSION\_PCT, and MANAGER.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER
100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	(null)	(null)
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	(null)	(null)
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	(null)	(null)
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	(null)
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	(null)
105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	4800	(null)	(null)
106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	4800	(null)	(null)
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200	(null)	(null)
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	(null)	(null)
109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	(null)	(null)
110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_ACCOUNT	8200	(null)	(null)
111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	7700	(null)	(null)





**Select EMPLOYEE\_ID, FIRST\_NAME, SALARY from EMPLOYEES**

Worksheet

Query Builder

Select EMPLOYEE\_ID, FIRST\_NAME, SALARY from EMPLOYEES

Query Result x

    SQL | Fetched 50 rows in 0.004 seconds

	EMPLOYEE_ID	FIRST_NAME	SALARY
1	100	Steven	24000
2	101	Neena	17000
3	102	Lex	17000
4	103	Alexander	9000
5	104	Bruce	6000
6	105	David	4800

**Select EMPLOYEE\_ID, FIRST\_NAME, SALARY from EMPLOYEES where salary > 2300**





Worksheet

Query builder

Select EMPLOYEE\_ID, FIRST\_NAME, SALARY from EMPLOYEES where salary>2300

▲▼

Query Result x

 SQL | Fetched 50 rows in 0.003 seconds

	EMPLOYEE_ID	FIRST_NAME	SALARY
1	100	Steven	24000
2	101	Neena	17000
3	102	Lex	17000


Select EMPLOYEE\_ID, FIRST\_NAME, SALARY from EMPLOYEES  
where salary >= 12000 and salary<=18000;

Worksheet

Query Builder

Select EMPLOYEE\_ID, FIRST\_NAME, SALARY from EMPLOYEES  
where salary >= 12000 and salary<=18000;

Query Result x

 | All Rows Fetched: 8 in 0.003 seconds

	EMPLOYEE_ID	FIRST_NAME	SALARY
1	101	Neena	17000
2	102	Lex	17000
3	108	Nancy	12008
4	145	John	14000
5	146	Karen	13500
6	147	Alberto	12000
7	201	Michael	13000
8	205	Shelley	12008

### III. 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 Airthmatic Operators

### Example

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

This example uses the addition operator to calculate a salary increase of \$300 for 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 Oracle server ignores blank spaces before and after the arithmetic operator.

### Output:

	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

### 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:**

Worksheet Query Builder

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

Query Result x

SQL | Fetched 50 rows in 0.029 seconds

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

**Example2 :**

Worksheet Query Builder

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

Query Result x

SQL | Fetched 50 rows in 0.005 seconds

	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  $(12 * \text{salary}) + 100$  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)

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

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

## 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.

### I. SQL Comparison Operators

Operator	Description
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

<b>!&lt;</b>	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.
<b>!&gt;</b>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.

## II. SQL Logical Operators

Operator	Description
<b>ALL</b>	The ALL operator is used to compare a value to all values in another value set.
<b>AND</b>	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause
<b>ANY</b>	The ANY operator is used to compare a value to any applicable value in the list as per the condition.
<b>BETWEEN</b>	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
<b>EXISTS</b>	The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
<b>IN</b>	The IN operator is used to compare a value to a list of literal values that have been specified.
<b>LIKE</b>	The LIKE operator is used to compare a value to similar values using wildcard operators.
<b>NOT</b>	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.
<b>OR</b>	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
<b>NULL</b>	The NULL operator is used to compare a value with a NULL value.
<b>UNIQUE</b>	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).