**Experiment No. 7**

|  |  |
| --- | --- |
| **Aim** | **Write Queries using Arithmetic operators.** |
| **Resources required** | |  |  | | --- | --- | | Sr. No. | **Instrument/ Object** | | 1 | Computer system with all necessary components like; motherboard, random access memory (RAM), read-only memory (ROM), internal hard disk drives, Mouse, Keyboard, and RDBMS applications such as Oracle Express Edition, MySQL, SQLite, Oracle Live SQL etc. | |
| **Procedure** | 1. Create tables for the given application 2. Apply Arithmetic operators on the given application |
| **Exercise** | 1. Consider the following schema Orders(cust\_id, order\_id, items, amount) Write queries for the following:   CREATE TABLE ORDERS1  (  CUST\_ID NUMBER(10),  ORDER\_ID NUMBER(10),  ITEMS VARCHAR(8),  AMOUNT NUMBER(7,2)  );    INSERT INTO ORDERS1 VALUES (201,651,'LAPTOP',78950.00);  INSERT INTO ORDERS1 VALUES (202,652,'MOBILE',75950.00);  INSERT INTO ORDERS1 VALUES (203,653,'TV',85950.00);     * 1. Display new column named total\_amount which is 200 added to the amount field.      1. Display new column named offer\_price which is 100 subtracted from the amount field.      * 1. Display new column named revised\_amount which is multiplied by 5 times the amount field.      * 1. Display new column named half\_amount which is divided by 2 to the amount field. |

**1.What is precedence/ SQL arithmetic order in arithmetic operators?**

In SQL, arithmetic operators follow a defined precedence or order of operations that

determines how expressions with multiple operators are evaluated. First and foremost,

any calculations enclosed within parentheses are executed before anything else, allowing

you to explicitly control the sequence of operations. After parentheses are handled, SQL

evaluates multiplication (\*) and division (/) operators next, processing them from left to

right as they appear in the expression. Only after these operations are completed does

SQL evaluate addition (+) and subtraction (-), also from left to right. This precedence

ensures that more complex arithmetic expressions are calculated correctly and

consistently, preventing ambiguity. For example, in the expression 10 + 5 \* 2, the

multiplication is performed first resulting in 5 \* 2 = 10, and then the addition takes

place, resulting in a final value of 20. If you want to change this default order, you can

use parentheses to group operations and force SQL to evaluate parts of the expression in a different sequence, such as (10 + 5) \* 2, which would first add 10 + 5 to get 15 and then multiply by 2 for a result of 30. Understanding the precedence of arithmetic

operators is essential for writing accurate SQL queries and avoiding unexpected results.

1. **Explain the SQL exponentiation operator.**

In SQL, the exponentiation operator is used to raise a number to the power of another

number, performing the mathematical operation of exponentiation. Unlike the common

arithmetic operators like addition or multiplication, SQL does not have a standard symbol

(such as ^ or \*\*) universally recognized across all database systems for exponentiation.

Instead, many SQL databases provide built-in functions, such as the POWER() function, to

perform exponentiation. The POWER(base, exponent) function takes two arguments –the base number and the exponent — and returns the base raised to the power of the

exponent. For example, POWER(2, 3) calculates 2 raised to the power of 3, which equals

8. This function is essential when performing calculations that involve powers or roots within SQL queries. Because the exponentiation operator or function might vary or not exist in some SQL dialects, it’s important to consult the documentation of the specific SQL database being used to know how to perform exponentiation correctly. Overall,exponentiation in SQL is typically done using functions like POWER() which provide a straightforward way to calculate powers in arithmetic expressions.

**3. List few compound operators.**

Compound operators are shorthand operators that combine an arithmetic operation with an

assignment, allowing you to update the value of a column or variable more concisely. Some common

compound operators include:

 += (Addition assignment)

 -= (Subtraction assignment)

 \*= (Multiplication assignment)

 /= (Division assignment)

 %= (Modulo assignment, in some SQL dialects)

For example, instead of writing SET salary = salary + 1000, you can use SET salary += 1000

in SQL dialects that support compound operators.

|  |  |  |  |
| --- | --- | --- | --- |
| **Marks Obtained** | | | **Dated signature of Teacher** |
| Process Related (15) | Product Related (35) | Total (50) |  |
|  |  |  |  |

**Experiment No. 8**

|  |  |
| --- | --- |
| **Aim** | **Apply built-in Logical operators on the given data** |
| **Resources required** | |  |  | | --- | --- | | Sr. No. | **Instrument/ Object** | | 1 | Computer system with all necessary components like; motherboard, random access memory (RAM), read-only memory (ROM), internal hard disk drives, Mouse, Keyboard, and RDBMS applications such as Oracle Express Edition, MySQL, SQLite, Oracle Live SQL etc. | |
| **Procedure** | 1. Create tables for the given application 2. Apply logical operators on the given application |
| **Exercise** | 1. Consider the following schema Emp(empno,ename,job,mgr,hiredate,sal,comm,deptno,city,) Write queries for the following:   CREATE TABLE EMP10  (  EMPNO NUMBER(10),  ENAME VARCHAR (10),  CITY VARCHAR(10),  JOB VARCHAR(10),  MGR NUMBER(8),  HIREDATE DATE,  SAL NUMBER(7,2),  COMM NUMBER(8,2),  DEPTNO NUMBER(5,0)  );  C:\Users\DELL\AppData\Local\Temp\ksohtml15436\wps1.jpg  INSERT INTO EMP10  VALUES(1,'VEDIKA','DELHI','TEACHER',200,'24DEC2015',9000,4562,10);  INSERT INTO EMP10  VALUES(2,'PRAJKTA','PUNE','CLERK',300,'12JAN2014',8000,4563,30);  INSERT INTO EMP10  VALUES(3,'HARSHAL','MUMBAI','ENGINEER',400,'12JUN2024',7000,4564,50);  INSERT INTO EMP10  VALUES(4,'NILKA','NAGPUR','DOCTOR',500,'12JUN2024',20000,4564,40);  INSERT INTO EMP10  VALUES(5,'KALYANI','NASHIK','MANAGER',700,'06JUL2023',30000,4565,50);  C:\Users\DELL\AppData\Local\Temp\ksohtml15436\wps2.jpg  i.Display employees whose city is ‘Mumbai’ and earns more than 50000  C:\Users\DELL\AppData\Local\Temp\ksohtml15436\wps3.jpg  ii.Display employees who job is Clerk or commission is 500  C:\Users\DELL\AppData\Local\Temp\ksohtml15436\wps4.jpg  iii.Display details of employees whose salary is between 20000 and 50000.  C:\Users\DELL\AppData\Local\Temp\ksohtml15436\wps5.jpg  iv.Display details of employees who stays at Mumbai, Pune, Nashik or Nagpur  C:\Users\DELL\AppData\Local\Temp\ksohtml15436\wps6.jpg |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Marks Obtained** | | | **Dated signature of Teacher** |
| Process Related (15) | Product Related (35) | Total (50) |  |
|  |  |  |  |

**Experiment No. 9**

|  |  |
| --- | --- |
| **Aim** | **Implement Relational operators to apply various conditions in a query.** |
| **Resources required** | |  |  | | --- | --- | | Sr. No. | **Instrument/ Object** | | 1 | Computer system with all necessary components like; motherboard, random access memory (RAM), read-only memory (ROM), internal hard disk drives, Mouse, Keyboard, and RDBMS applications such as Oracle Express Edition, MySQL, SQLite, Oracle Live SQL etc. | |
| **Procedure** | 1. Create tables for the given application 2. Apply Relational operators on the given application |
| **Exercise** | 1. Consider the following schema Student (stu\_name, course\_id, Roll\_no, percentage) Write queries for the following:   CREATE TABLE Student1 (  stu\_name VARCHAR(100),  course\_id number(20),  Roll\_no number(10) PRIMARY KEY,  percentage number(5,2)  );  Insert values student1(‘vedika’,201,1,70.000);  Insert values student1(‘kalyani’,121,10,58.000);  Insert values student1(‘prajakta’,122,16,95.000);  Insert values student1(‘harshal’,123,18,62.000);  Insert values student1(‘arya’,125,14,55.000);  Insert values student1(‘diksha’,124,20,88.000);  select\*from student1;     1. Select stu\_name, course\_id, from Student WHERE percentage is >=60 and <=100;   SELECT stu\_name, course\_id  FROM Student1  WHERE percentage BETWEEN 60 AND 100;     1. Select details of students whose Roll numbers are above 15;   SELECT \*  FROM Student1  WHERE Roll\_no > 15;     1. Select stu\_id, Roll\_no from Student WHERE course\_id! =121;   SELECT stu\_name, Roll\_no  FROM Student1  WHERE course\_id != 121; |

|  |  |  |  |
| --- | --- | --- | --- |
| **Marks Obtained** | | | **Dated signature of Teacher** |
| Process Related (15) | Product Related (35) | Total (50) |  |
|  |  |  |  |

**Experiment No. 10**

|  |  |
| --- | --- |
| **Aim** | **Use Set operators to perform different operations** |
| **Resources required** | |  |  | | --- | --- | | Sr. No. | **Instrument/ Object** | | 1 | Computer system with all necessary components like; motherboard, random access memory (RAM), read-only memory (ROM), internal hard disk drives, Mouse, Keyboard, and RDBMS applications such as Oracle Express Edition, MySQL, SQLite, Oracle Live SQL etc. | |
| **Procedure** | 1. Create tables for the given application 2. Apply set operators on the given tables |
| **Exercise** | * + - 1. Consider following Schema : emp1(empno,ename,deptno) emp2(empno,ename,deptno)   CREATE TABLE emp1 (  empno number(10) PRIMARY KEY,  ename VARCHAR(100),  deptno number(10)  );  CREATE TABLE emp2 (  empno number(10) PRIMARY KEY,  ename VARCHAR(100),  deptno number(10)  );  INSERT INTO emp1 values(101, 'vedika', 10);  INSERT INTO emp1 values(102, 'prajkta', 20);  INSERT INTO emp1 values(103, 'kalyani', null);  INSERT INTO emp1 values(104, 'vrushal', 30);  INSERT INTO emp1 values(105, 'harshal',null);  select\*from emp1;    INSERT INTO emp2 values(201, 'vedika', 40);  INSERT INTO emp2 values(202, 'shreya', null);  INSERT INTO emp2 values(203, 'arya', 30);  INSERT INTO emp2 values(204, 'harshal', 50);  INSERT INTO emp2 values(205, 'vivek',null);  select\*from emp2;    Write SQL commands for the following statements.   1. Display the names of employees including duplicate employee names.   SELECT ename FROM emp1  UNION ALL  SELECT ename FROM emp2;       1. Display the names of employees excluding duplicate employee names.   SELECT ename FROM emp1  UNION  SELECT ename FROM emp2;     1. Display the common employee names from both the tables.   SELECT ename FROM emp1  INTERSECT  SELECT ename FROM emp2;     1. List employees who are not assigned to any department   SELECT empno, ename FROM emp1  WHERE deptno IS NULL  UNION  SELECT empno, ename FROM emp2  WHERE deptno IS NULL; |

|  |  |  |  |
| --- | --- | --- | --- |
| **Marks Obtained** | | | **Dated signature of Teacher** |
| Process Related (15) | Product Related (35) | Total (50) |  |
|  |  |  |  |