|  |
| --- |
| **EXP 1** **CREATING AND MANAGING TABLES** |

**AIM:**

To use data definition language commands and define the schema of the database.

**DESCRIPTION:**

A **Data Definition Language** (**DDL**) is a computer language for defining data structures. The term was first introduced in relation to the [Codasyl](http://en.wikipedia.org/wiki/Codasyl) database model, where the schema of the database was written in a Data Definition Language describing the records, fields, and "sets" making up the user Data Model. Initially it referred to a subset of SQL, but is now used in a generic sense to refer to any formal language for describing data or information structures.

**SQL**

Initially, DDL was a subset of SQL statements. perhaps the most common code is CREATE TABLE code in this kind of statements.

**CREATE statements**

**Create** - To make a new database, table, index, or stored query. A CREATE statement in [SQL](http://en.wikipedia.org/wiki/SQL) creates an object inside of a relational database management system (RDBMS). The types of objects that can be created depends on which RDBMS is being used, but most support the creation of tables, indexes, users, synonyms and [databases](http://en.wikipedia.org/wiki/Database). Some systems (such as [PostgreSQL](http://en.wikipedia.org/wiki/PostgreSQL)) allow CREATE, and other DDL commands, inside of a transaction and thus they may be rolled back.

**CREATE TABLE statement**

Perhaps the most common CREATE command is the CREATE TABLE command. The typical usage is:

CREATE [TEMPORARY] TABLE *[table name]* ( *[column definitions]* ) *[table parameters]*.

For example, the command to create a table named **employees** with a few sample columns would be:

CREATE TABLE employees (

id INTEGER PRIMARY KEY,

first\_name CHAR(50) NULL,

last\_name CHAR(75) NOT NULL,

dateofbirth DATE NULL

);

**DROP statements**

Drop - To destroy an existing database, table, index, or view.

The typical usage is simply DROP *objecttype* *objectname*. For example, the command to drop a table named **employees** would be:

DROP TABLE employees;

The DROP statement is distinct from the [DELETE](http://en.wikipedia.org/wiki/Delete_%28SQL%29) and (non-standard) [TRUNCATE](http://en.wikipedia.org/wiki/Truncate_%28SQL%29) statements, in that they do not remove the table itself. For example, a DELETE statement might delete some (or all) data from a table while leaving the table itself in the database, whereas a DROP statement would remove the entire table from the database.

**ALTER statements**

Alter - To modify an existing database object.

The typical usage is ALTER *objecttype* *objectname* *parameters*. For example, the command to add (then remove) a column named **bubbles** for an existing table named **sink** would be:

ALTER TABLE sink ADD bubbles INTEGER;

ALTER TABLE sink DROP COLUMN bubbles;

**Sample query**

**1.Create student table with the following attributes Reg No, Name , Email\_id**

Create table student(Reg\_No number(3),Name varchar(10),email\_id varchar(20));

**2.Write a query to Add age and dept\_no to the student table**

Alter table student add(Age number(2),dept\_no number(5));

**3.Write a query to Rename the table name to Student2010**

Rename student to student2010;

**4.Change the name of attribute reg\_no to student\_no**

Alter table student2010 rename column reg\_no to student\_no;

**5.Make the attribute reg\_no to primary key**

Alter table student2010 add constraint C1 primary key(student\_no);

**Exercise:**

6)Drop the attribute age from the table student2010.

7)Describe the table student2010.

8) Drop table student2010.

|  |
| --- |
| **EXP NO:**2 **BASIC SQL** |

**AIM:**

To execute the given queries using the data manipulation language commands.

**DESCRIPTION:**

DML:

Data Manipulation Language (DML) statements are used to define the data in the database

INSERT – to insert data into tables.

INSERT INTO *table* [(*column* [*, column...*])]

VALUES *(value* [*, value...*]);

UPDATE -- to update data in the database.

UPDATE *table*

SET *column* = *value* [, *column* = *value, ...*]

[WHERE *condition*];

DELETE – to delete data from the database.

DELETE [FROM] *table*

[WHERE *condition*];

SELECT – to retrieve the data from the database.

SELECT \*|{[DISTINCT] *column*|*expression* [*alias*],...}

FROM *table;*

SELECT \*|{[DISTINCT] *column|expression* [*alias*],...}

FROM *table*

[WHERE *condition(s)*];

A where clause is used to restrict the rows returned.

**OPERATORS USED IN WHERE CLAUSE:**

1. **< >**

Greater than, less than

Used to filter the data with the specified condition

Ex. Where salary > 1500

1. **And / Or**

and, or operators is used to check more than one conditions.

And:- retrives the row when both the conditions satisfy.

Or:- retrives the row when atleast one condition satisfy.

1. **=, !=, <>**

Equal( = ) and not equal( != , <> )

Used to filter data with specified condition.

1. Null, not null

Check whether the data in a field is null or not null.

Ex: where deptno is null;

1. **In, not in**

Filters the data when the conditions satisfy.

Ex: where deptno in(10, 20, 30);

1. **Like**

Like is used for string formats comparisons.

Ex: where empname like ‘s%’;

1. **Between and**

Used to compare the values present in a range.

Ex: where age between 20 and 30;

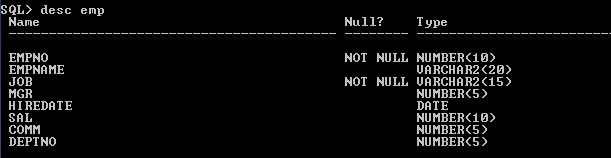
**TABLES:**

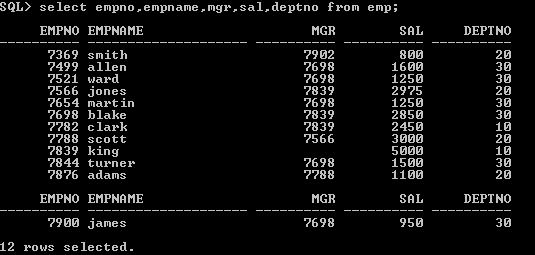
DEPT TABLE:

Text

Description automatically generated

EMP TABLE:





**Sample Query**

**1) List the employees belong to departments analyst or salesman.**

Select ename from emp where job in (‘analyst’,’salesman’);

**2) List details of employees whose salary is greater than 1500**

Select ename from emp whose sal>1500;

**3) List the employee names who are not earning commission.**

Select ename from emp where comm is null;

**4) List employee names with employee number in 7369,7521,7893,7934,7788.**

Select ename from emp where empno in (7369,7521,7893,7934,7788);

**Exercise**

5) List the employees not in department number 10,30,40

6) List the different destinations in a company.

7) List the name of employee who earns commission.

8) List the name of employees who are not assigned to any of the department.

9) List the name of employees whose name starts or ends with ‘s’.

10) List the name of employees whose name is 4 characters.

11) List the name of employee with names containing I in second letter.

|  |
| --- |
| **EXP NO:** 3 **ADVANCED SQL** |

**AIM:**

To execute the given commands making use of aggregate functions, group by clause and order by clause.

**DESCRIPTION:**

**AGGREGATE FUNCTIONS:**

They take a collection of values and return a single value as result. The various aggregate functions are:-

1)Sum: - takes a collection of values and returns the sum of the values.

Eg: select sum(sal) from emp;

2)Avg:- returns the average of the values.

Eg: select avg(sal) from emp;

3)Count:- returns the number of elements in the collection.

Eg: select count(\*) from emp;

4)Min:- returns the minimum value in a collection.

Eg: select min(sal) from emp;

5)Max:- returns the maximum value in a collection.

Eg: select max(sal) from emp;

The input to sum and average must be a collection of numbers, but the other operators can operate on collections of non-numeric data types, such as strings as well. The average function will return the average of the given tuple.

The aggregation function count is used frequently to count the number of tuples in relation .The for this function in SQL is count (\*).

**Distinct Keyword**

To eliminate the duplicates the keyword distinct is used in the aggregation expression. Sql does not allow the use of distinct with count(\*).It is legal to use distinct with max and min, even though the result does not change.

Eg: Select count (distinct job);

**GROUP BY Clause:-**

To apply aggregate function to a group of sets of tuples. The attributes given in the group by clause are used to form groups. Tuples with some value on all attributes in the group by clause are placed in one group.

Eg: Select branch\_name,avg(bal) from account group by branch\_name;

**ORDER BY Clause:-**

This clause causes the tuples in the result of a query to appear in sorted order. We specify asc for ascending order and desc for descending order.

Eg: Select \* from loan order by amount desc;

**EXERCISE:**

1.List the number of employees working with the company.

2.List the number of designations available in the emp table

3.List the total salaries paid to the employees.

4.List the maximum, minimum and average salary in the company.

5.List the maximum salary paid to a sales man.

6.List the total salary, maximum and minimum and average salary of employees jobwise, for department 20

7.List the number of employees and average salary for employees in department 20.

8.List name, salary and pf amount of all employees.

9.List names of employees who are more than 25 years old in the company

10.List the employee details in the ascending order of their basic salary.

11.List the employee name and hire date in the descending order of the hire date.

12.List employee name, salary, pf, hra, da and gross, order the results in the ascending order of gross. Hra is 50% of the salary and da is 30% of the salary.

13.List the department numbers and number of employees in each department.

14.List the department number and total salary payable in each department.

15.List the jobs and number of employees in each job. The result should be in the descending

order of the number of employees.

16.List the total salary, maximum and minimum salary and average salary of the employees jobwise.

17.List the total salary, maximum and minimum salary and average salary of the employees, for department 20.

18.List the average salary of the employees job wise, for department 20 and display only those rows having an average salary > 1000

**RESULT:**

Thus all the aggregation and the group by queries are studied and the questions are done and the output is verified successful

|  |
| --- |
| **EXP NO:**4 **SET OPERATIONS AND JOINS** |

**AIM:**

To execute the given queries using set operators and joins.

**DESCRIPTION:**

**Set Operators**

The three *set operators* union, intersect and minus allow to serially combine more than one select statements. Although more than one select statement will then be present, only *one* result set is then returned.

The following list briefly describes the three set operations supported by Oracle SQL:

**1) UNION**

union all is very similar to union, however, it dismisses duplicate rows found across different select statements:

select col\_1, col\_2, col\_3 from table\_1 **union**

select col\_1, col\_2, col\_3 from table\_2;

**2) INTERSECT**

intersect only returns the rows that are found in all select statements:

select col\_1, col\_2, col\_3 from table\_1 **intersect**

select col\_1, col\_2, col\_3 from table\_2;

**3) MINUS**

minus returns all rows from the first select statements except those who are duplicated in a following select statement:

select col\_1, col\_2, col\_3 from table\_1 **minus**

select col\_1, col\_2, col\_3 from table\_2;

**SQL JOIN**

The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.

Tables in a database are often related to each other with keys.

**Different SQL JOINs**

Before we continue with examples, we will list the types of JOIN you can use, and the differences between them.

**JOIN**: Return rows when there is at least one match in both tables

**LEFT JOIN**: Return all rows from the left table, even if there are no matches in the right table

**RIGHT JOIN**: Return all rows from the right table, even if there are no matches in the left table

**FULL JOIN**: Return rows when there is a match in one of the tables

**SQL INNER JOIN Keyword**

The INNER JOIN keyword return rows when there is at least one match in both tables.

**Syntax**

|  |
| --- |
| SELECT column\_name(s) FROM table\_name1 INNER JOIN table\_name2 ON table\_name1.column\_name=table\_name2.column\_name |

**PS:** INNER JOIN is the same as JOIN.

**SQL LEFT JOIN Keyword**

The LEFT JOIN keyword returns all rows from the left table (table\_name1), even if there are no matches in the right table (table\_name2).

**Syntax**

|  |
| --- |
| SELECT column\_name(s) FROM table\_name1 LEFT OUTER JOIN table\_name2 ON table\_name1.column\_name=table\_name2.column\_name |

**SQL RIGHT JOIN Keyword**

The RIGHT JOIN keyword Return all rows from the right table (table\_name2), even if there are no matches in the left table (table\_name1).

**Syntax**

|  |
| --- |
| SELECT column\_name(s) FROM table\_name1 RIGHT OUTER JOIN table\_name2 ON table\_name1.column\_name=table\_name2.column\_name |

**SQL FULL JOIN Keyword**

The FULL JOIN keyword return rows when there is a match in one of the tables.

**Syntax**

|  |
| --- |
| SELECT column\_name(s) FROM table\_name1 FULL OUTER JOIN table\_name2 ON table\_name1.column\_name=table\_name2.column\_name |

**Exercise:**

**Sets:**

1.Display all the names of the employees who is working in this organization without repeat.

2 .Display the names of the employee who does not have their dependence working on the same organisation.

3.Display the name of the employee who is dependent and working as staff.

4. Display the name of the employee who is dependent but not working as a staff.

**Joins:**

1. Display the employee name ,employee id ,dept name and location name using a joint condition.

2.Display all names of the employees even there is no matching departement.

3.Display all the departement names even if they are not assigned for a employee.

4.Retrive all rows of employee table even if there is no matching departement for employee.

5. Display all the departement number if does not have any matching location id.

6.Display all departemnet name and location name even if they do not have matching

location id.

|  |
| --- |
| **EX.NO.5**  **SUBQUERIES** |

**AIM:**

To execute given query using the concept of subqueries.

**DESCRIPTION:**

Subquery is usually added in the WHERE Clause of the sql statement. Most of the time, a subquery is used when you know how to search for a value using a SELECT statement, but do not know the exact value. Subqueries are an alternate way of returning data from multiple tables.

Subqueries can be used with the following sql statements along with the comparision operators like =, <, >, >=, <= etc.

* Update
* Insert
* Select
* Delete

**DETAILED PROCEDURE**

1) Usually, a subquery should return only one record, but sometimes it can also return multiple records when used with operators like IN, NOT IN in the where clause. The query would be like,

SELECT first\_name, last\_name, subject FROM student\_details   
WHERE games NOT IN ('Cricket', 'Football');

The output would be similar to:

|  |  |  |
| --- | --- | --- |
| first\_name | last\_name | subject |
| Shekar | Gowda | Badminton |
| Priya | Chandra | Chess |

2) Lets consider the student\_details table which we have used earlier. If you know the name of the students who are studying science subject, you can get their id's by using this query below,

SELECT id, first\_name FROM student\_details   
WHERE first\_name IN ('Rahul', 'Stephen');

but, if you do not know their names, then to get their id's you need to write the query in this manner,

SELECT id, first\_name FROM student\_details   
WHERE first\_name IN (SELECT first\_name   
FROM student\_details WHERE subject= 'Science');

|  |  |
| --- | --- |
| Id | first\_name |
| 100 | Rahul |
| 102 | Stephen |

In the above sql statement, first the inner query is processed first and then the outer query is processed.

3) Subquery can be used with INSERT statement to add rows of data from one or more tables to another table. Lets try to group all the students who study Maths in a table 'maths\_group'.

INSERT INTO maths\_group(id, name) SELECT id, first\_name || ' ' || last\_name   
FROM student\_details WHERE subject= 'Maths'

4) A subquery can be used in the SELECT statement as follows. Lets use the product and order\_items table defined in the sql\_joins section.

select p.product\_name, p.supplier\_name, (select order\_id from order\_items where product\_id = 101) as order\_id from product p where p.product\_id = 101

|  |  |  |
| --- | --- | --- |
| product\_name | supplier\_name | order\_id |
| Television | Onida | 5103 |

**Sample Output/Input**

1. SQL> Select Member.id,Member.name,Member.address,Member.city,Member.state,

Member.pin from Member,Catalog,Order\_summer,Order\_detail

where(Catalog.Book\_id=Order\_detail.Book\_id and Order\_Summer.Order\_no=Order\_Detail.Order\_no and Order\_Summer.Member\_id=Member.id and Catalog.Title='DBMS');

1. Get the details of the titles whose price is greater than the average price and whose year of publishing is greater than the average year of publishing.

SQL> Select \* from Catalog where Price> (select avg(Price) from catalog) and Year> (select avg(Year) from catalog);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| BOOK\_ID | TITLE | AUTHOR\_ID | PUBLISH\_ID | CATEGORY\_ID | YEAR | PRICE |
| 5555 | CA | 5678 | 5656 | 787 | 2010 | 3500 |

1. Get the title, year, and price of all the books in the ascending order of the year of publishing for which an order is placed.

SQL> Select distinct title,Year,Price from Catalog,Order\_Detail where Order\_Detail.Book\_id=Catalog.Book\_id order by Year asc;

|  |  |  |
| --- | --- | --- |
| **TITLE** | **YEAR** | **PRICE** |
| DBMS | 1990 | 2000 |
| OS | 1995 | 500 |
| JAVA | 2004 | 750 |

1. Find out how many order has been placed for DBMS book.

SQL> Select sum(Quantity) from Order\_Detail,Catalog where Order\_Detail.Book\_id=Catalog.Book\_id and Title='DBMS';

|  |
| --- |
| **SUM(QUANTITY)** |
| 5 |

1. Find out the order status for book ordered by Ravi.

SQL> Select Order\_Status from Order\_Summer where Member\_id in(Select Id from Member where Name='RAVI');

|  |
| --- |
| **ORDER\_STATUS** |
| PROCESSED |
| PROCESSING |

**Additional Questions:**

1. Display the total price for all products.
2. Display the total price only for office chair and table.
3. Display the details of order placed by customer john Doe.
4. How many orders are placed by John Doe.
5. Find the no. of orders placed by each customer.
6. Find out the no. of orders placed on each date.
7. Find out how many products are ordered in each order.
8. Find out how many orders are placed for office chair product
9. Sort the order table based on the order date.
10. Find out the total quantity ordered for each product.
11. Display the products for which more than 3 quantity are orders and sort the result on the sum of quantity ordered.
12. Display the details of the customer how ordered the product whose price is the maximum among all the products
13. Find out the details of customer who has not yet placed an order.
14. Find out the details of products which are not ordered by any customer.
15. Find out the details of customer who placed order for office chair.
16. Give a 10% increase to all the product price and display the column with a name updated price and sort the table with that column.
17. Find out the total no of orders placed by each customer, display the customer who have placed more than 3 orders exclude customer 1 from the result.
18. Find the no. of orders placed by the customer belonging to newyork city.
19. Find the product which has sold the maximum.
20. Find the products whose price is greater than the average price of chair and table.

|  |
| --- |
| **Ex.No.6** **CORRELATED SUBQUERY** |

**AIM**

To perform the given queries using correlated subquery.

**DESCRIPTION:**

A **correlated sub-query** is a term used for specific types of queries in SQL in computer databases. It is a sub-query (a query nested inside another query) that uses values from the outer query in its WHERE clause. The sub-query is evaluated once for each row processed by the outer query.

**DETAILED PROCEDURE:**

Here is an example for a typical correlated sub-query. In this example we are finding the list of employees (employee number and names) having more salary than the average salary of all employees in that employee's department.

SELECT employee\_number, name FROM employee AS e1

WHERE salary > (SELECT avg(salary)FROM employee

WHERE department = e1.department);

In the above query the outer query is,

SELECT employee\_number, name FROM employee AS e1

WHERE salary >

And the inner query is,

(SELECT avg(salary)

FROM employee

WHERE department = e1.department);

In the above nested query the inner query has to be executed for every employee as the department will change for every row. Hence the average salary will also change. The effect of correlated sub-queries can also be obtained using outer Joins.

**Sample Input and Output:**

Tables:-

Create the following tables

Create table customer(cust\_id number(3), cust\_name varchar2(10), annual\_revenue number(10), cust\_type varchar2(12));

Commit;

Create table SHIPMENT (SHIPMENT\_# number(3), CUST\_ID number(3) , WEIGHT number(4,1), TRUCK\_# number(3), DESTINATION varchar2(15),

SHIP\_DATE date);

Create table TRUCK (TRUCK\_# number(3), DRIVER\_NAME varchar2(15));

Create table CITY (CITY\_NAME varchar2(10), POPULATION number(15));

Solve the following complex SQL Queries using Exists or NOT Exists

1. What are the names of customers who have sent packages (shipments) to Sioux City?

CUST\_NAME

----------

Revathi

1. To what destinations have companies with revenue less than $1 million sent packages?

DESTINATION

---------------

London

Los Angeles

Sioux City

1. What are the names and populations of cities that have received shipments weighing over 100 pounds?

|  |  |
| --- | --- |
| **CITY\_NAME** | **POPULATION** |
| London | 100000000 |
| Manhattan | 10000000 |
| Rome | 200000000 |
| Sioux City | 5000000000 |

4.Who are the customers having over $5 million in annual revenue who have

sent shipments weighing less than 1 pound?

|  |
| --- |
| **CUST\_ID** |
| 311 |

5.Who are the customers having over $5 million in annual revenue who have

sent shipments weighing less than 1 pound or have sent a shipment to San

Francisco?

No row Selected

**Additional Questions:**

1. Who are the drivers who have delivered shipments for customers with annual revenue over $20 million to cities with populations over 1 million?
2. List the cities that have received shipments from customers having over $15 million in annual revenue.
3. List the names of drivers who have delivered shipments weighing over 100 pounds taken cumulatively.
4. List the name and annual revenue of customers who have sent shipments weighing over 100 pounds.
5. List the name and annual revenue of customers whose shipments have been delivered by truck driver named Jensen.
6. List customers who had shipments delivered by every truck. ( use NOT

EXISTS)

1. List cities that have received shipments from every customer. ( use NOT

EXISTS)

1. List drivers who have delivered shipments to every city. (use NOT EXISTS)
2. List Customers who are manufacturers or have sent a package to St. Louis.
3. List Cities of population over 1 million which have received a 100-pound

package from customer 311.