

1. Schema for the Practicals :

The overall design of the database is called the database schema. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an instance of a database schema.

Here, for the practicals, we are using schema for Banking Enterprise. This schema shown in below figure...

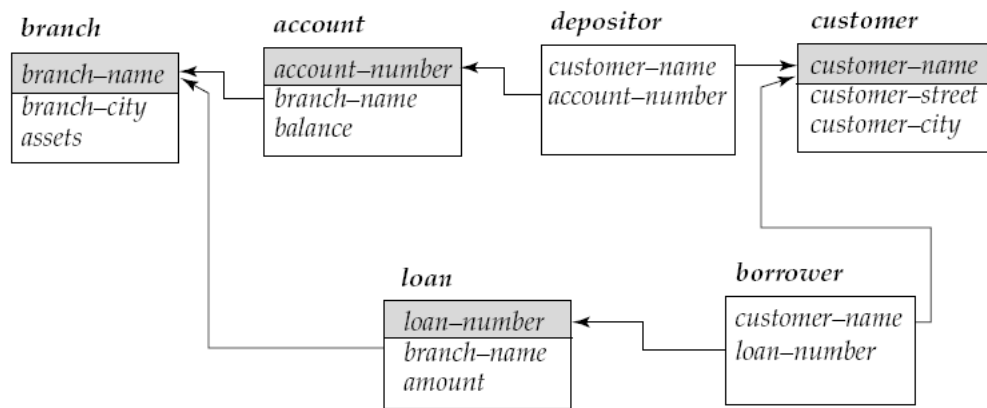


Figure 1: Banking Enterprise - A Schema

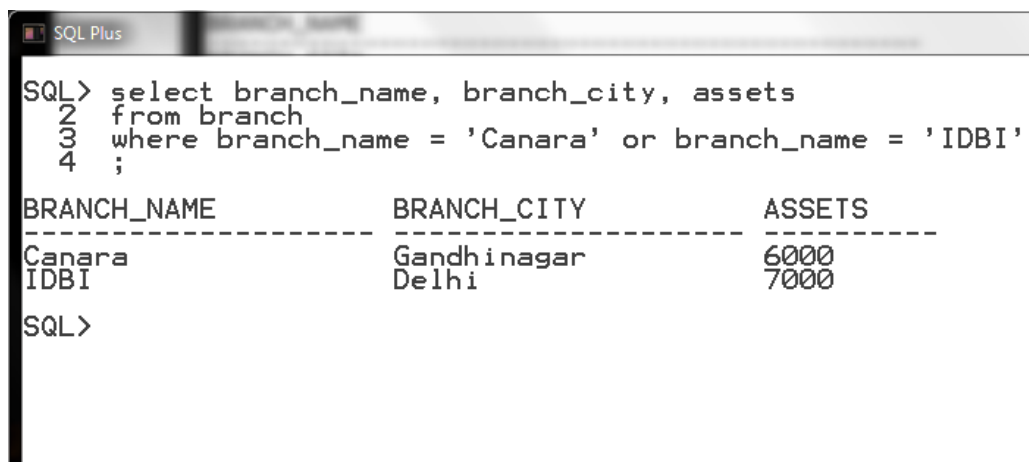
1. **Aim : To perform various SQL commands with logical Operators and Predicates.**

Logical Operator : There are three Logical Operators namely, AND, OR, and NOT. These operators compare two conditions at a time to determine whether a row can be selected for the output. When retrieving data using a SELECT statement, you can use logical operators in the WHERE clause, which allows you to combine more than one condition.

”OR” Logical Operator : If you want to select rows that satisfy at least one of the given conditions, you can use the logical operator, OR. **Syntax :**

```
SELECT C1,C2,...,Cn
FROM TABEL_NAME
WHERE <condition> OR <condition>
;
```

Here, in below figure show implemented ”OR Logical Operator”.



```
SQL> select branch_name, branch_city, assets
2   from branch
3   where branch_name = 'Canara' or branch_name = 'IDBI'
4   ;
```

BRANCH_NAME	BRANCH_CITY	ASSETS
Canara	Gandhinagar	6000
IDBI	Delhi	7000

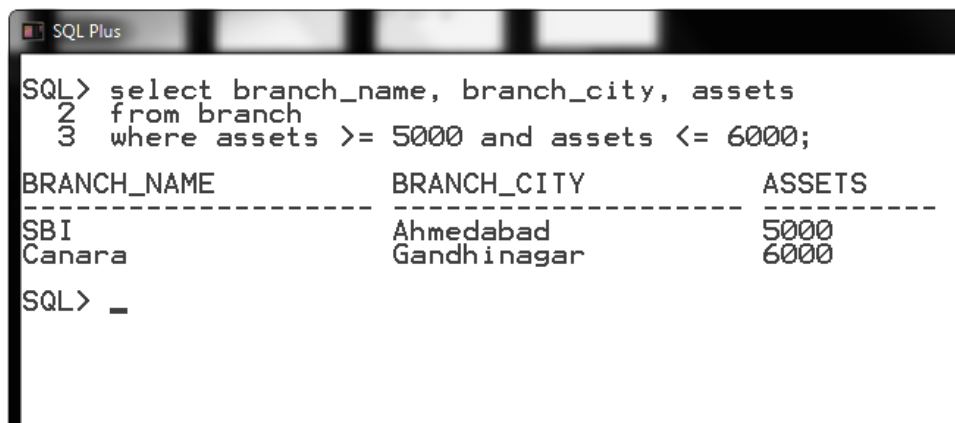
```
SQL>
```

Figure 2: Logical Operator : OR

”AND” Logical Operator : If you want to select rows that must satisfy all the given conditions, you can use the logical operator, AND. **Syntax :**

```
SELECT C1,C2,...,Cn
FROM TABEL_NAME
WHERE <condition> AND <condition>
;
```

Here, in below figure show implemented "AND Logical Operator".



```
SQL> select branch_name, branch_city, assets
2   from branch
3  where assets >= 5000 and assets <= 6000;
```

BRANCH_NAME	BRANCH_CITY	ASSETS
SBI	Ahmedabad	5000
Canara	Gandhinagar	6000

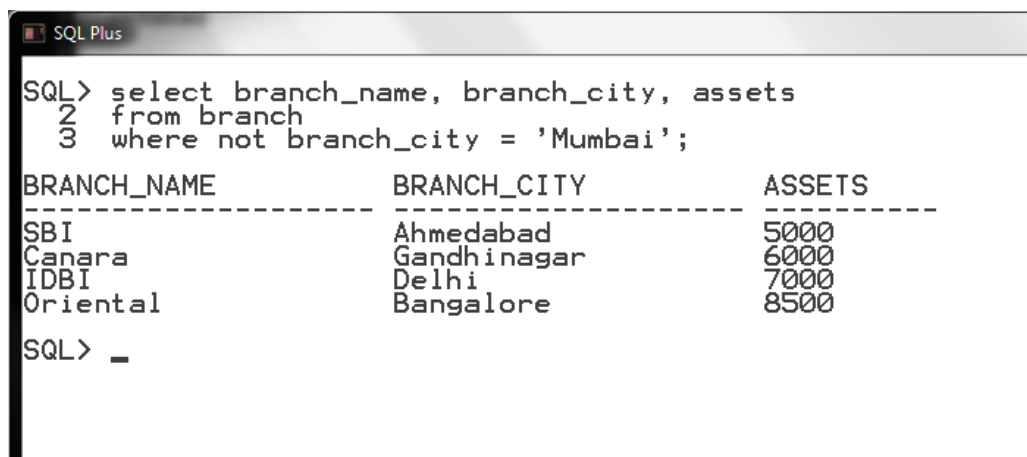
```
SQL> _
```

Figure 3: Logical Operator : AND

"NOT" Logical Operator: If you want to find rows that do not satisfy a condition, you can use the logical operator, NOT. NOT results in the reverse of a condition. That is, if a condition is satisfied, then the row is not returned. **Syntax** :

```
SELECT C1,C2,...,Cn
FROM TABEL_NAME
WHERE <condition> NOT <condition>
;
```

Here, in below figure show implemented "NOT Logical Operator".



```
SQL> select branch_name, branch_city, assets
2   from branch
3  where not branch_city = 'Mumbai';
```

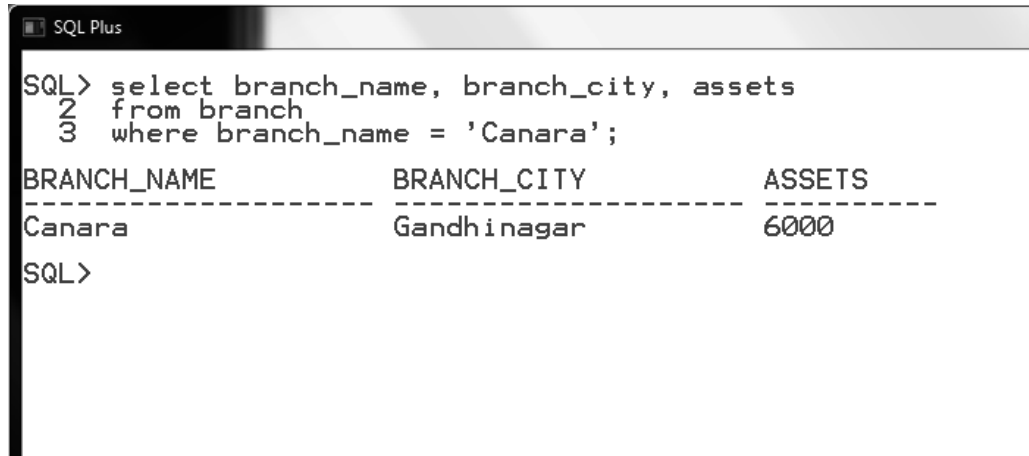
BRANCH_NAME	BRANCH_CITY	ASSETS
SBI	Ahmedabad	5000
Canara	Gandhinagar	6000
IDBI	Delhi	7000
Oriental	Bangalore	8500

```
SQL> _
```

Figure 4: Logical Operator : NOT

SQL Predicates : There are other comparison keywords available in sql which are used to enhance the search capabilities of a sql query. They are "WHERE","IN", "BETWEEN...AND", "IS NULL", "LIKE"...etc.

Here, in below figure show implemented "WHERE predicate".



```
SQL> select branch_name, branch_city, assets
2   from branch
3  where branch_name = 'Canara';
```

BRANCH_NAME	BRANCH_CITY	ASSETS
Canara	Gandhinagar	6000

```
SQL>
```

Figure 5: Predicate : WHERE

2. To Perform various Data Constraint with table level and column level specifications.

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be **column level** or **table level**. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

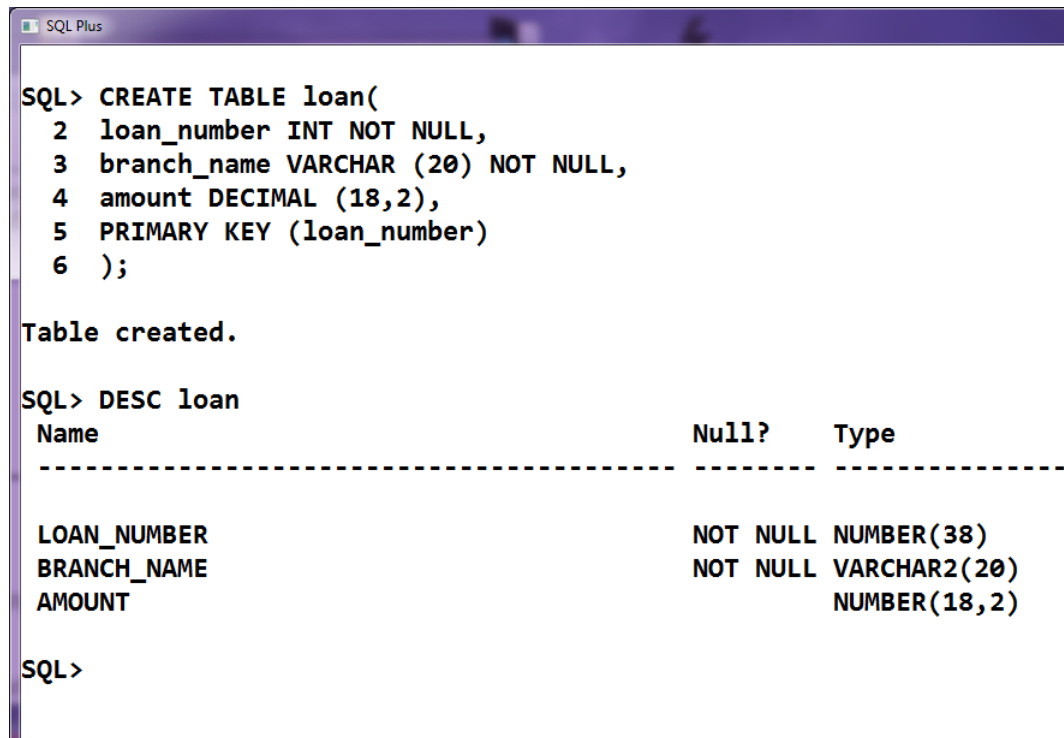
List of Constraints :

- Primary Key
- Foreign Key
- Unique Key
- Null Value

On the next page, we show the Primary Key constraint syntax at column level and table level with it's demo.

Syntax : Primary Key at table level

```
PRIMARY KEY ( <ColumnName>, ..., <ColumnName> )
```

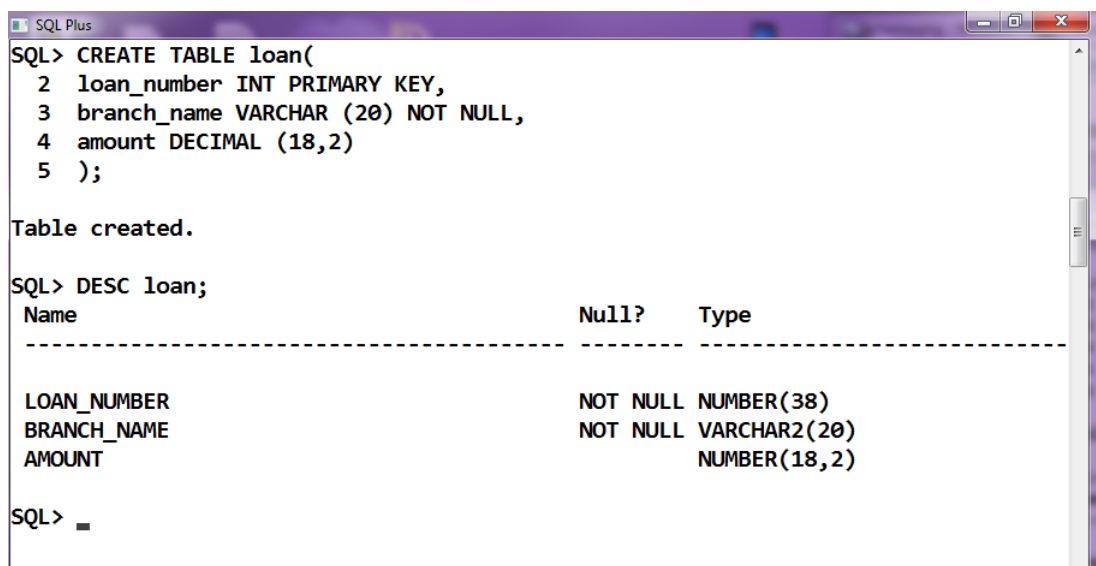


```
SQL> CREATE TABLE loan(  
2  loan_number INT NOT NULL,  
3  branch_name VARCHAR (20) NOT NULL,  
4  amount DECIMAL (18,2),  
5  PRIMARY KEY (loan_number)  
6  );  
  
Table created.  
  
SQL> DESC loan  
Name Null? Type  
-----  
LOAN_NUMBER NOT NULL NUMBER(38)  
BRANCH_NAME NOT NULL VARCHAR2(20)  
AMOUNT NUMBER(18,2)  
  
SQL>
```

Figure 6: Constraint at Table Level : PRIMARY KEY

Syntax : Primary Key at column level

```
<ColumnName> <Datatype> (<size>) PRIMARY KEY
```



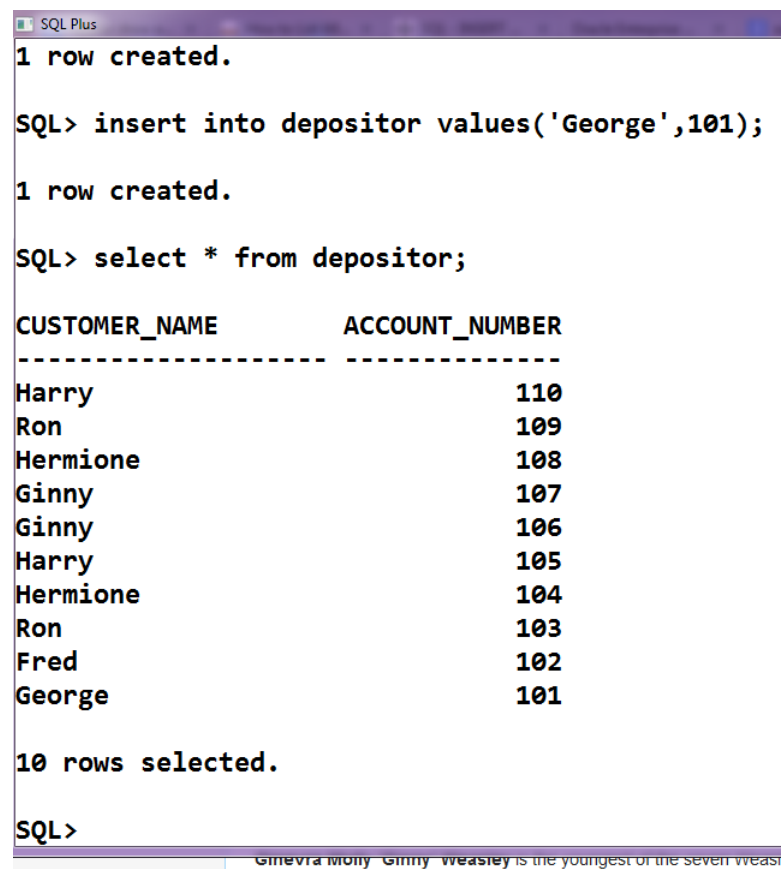
```
SQL> CREATE TABLE loan(  
2  loan_number INT PRIMARY KEY,  
3  branch_name VARCHAR (20) NOT NULL,  
4  amount DECIMAL (18,2)  
5  );  
  
Table created.  
  
SQL> DESC loan;  
Name Null? Type  
-----  
LOAN_NUMBER NOT NULL NUMBER(38)  
BRANCH_NAME NOT NULL VARCHAR2(20)  
AMOUNT NUMBER(18,2)  
  
SQL>
```

Figure 7: Constraint at Column Level : PRIMARY KEY

3. **To perform the concept of Grouping Data from tables in SQL.** The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. **Syntax : GROUP BY Clause**

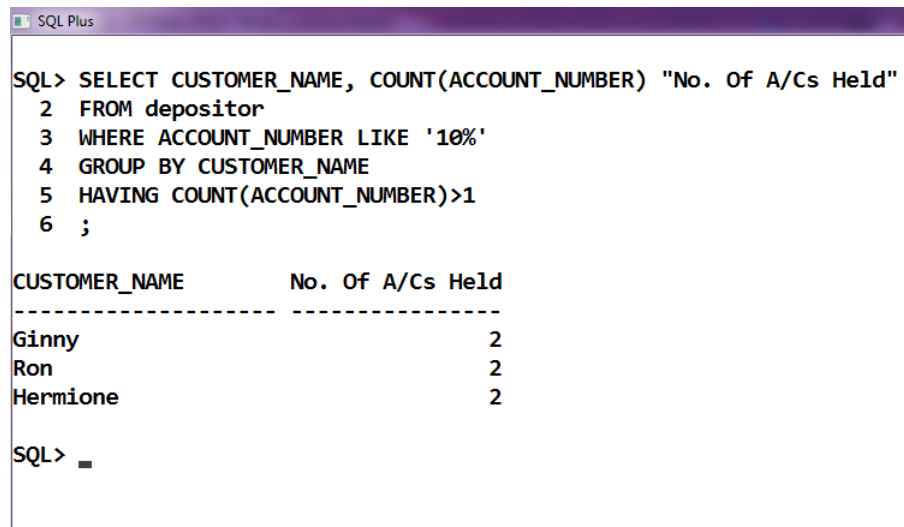
```
SELECT <column1>,<column2>,...,<columnN>,  
FUNCTION(<Expration>)  
FROM TABLE_NAME  
WHERE <Condition>  
GROUP BY <column1>,<column2>,...,<columnN>  
Other Clause  
;
```

Explanation : Here, in the demo we are grouping the customer name, whose account number like '10x' and customer have more then 1 account. First figure show the depositor table and second figure show the result of successful grouping.



```
SQL Plus  
1 row created.  
  
SQL> insert into depositor values('George',101);  
  
1 row created.  
  
SQL> select * from depositor;  
  
CUSTOMER_NAME      ACCOUNT_NUMBER  
-----  
Harry              110  
Ron                 109  
Hermione            108  
Ginny               107  
Ginny               106  
Harry               105  
Hermione            104  
Ron                 103  
Fred                102  
George              101  
  
10 rows selected.  
  
SQL>
```

Figure 8: Show the 'depositor' Table



The screenshot shows a terminal window titled "SQL Plus". It contains an SQL query and its output. The query is:
SQL> SELECT CUSTOMER_NAME, COUNT(ACCOUNT_NUMBER) "No. Of A/Cs Held"
2 FROM depositor
3 WHERE ACCOUNT_NUMBER LIKE '10%'
4 GROUP BY CUSTOMER_NAME
5 HAVING COUNT(ACCOUNT_NUMBER)>1
6 ;
The output is a table with two columns: "CUSTOMER_NAME" and "No. Of A/Cs Held". The data rows are: Ginny (2), Ron (2), and Hermione (2). The prompt "SQL> " is followed by a cursor.

```
SQL> SELECT CUSTOMER_NAME, COUNT(ACCOUNT_NUMBER) "No. Of A/Cs Held"  
2 FROM depositor  
3 WHERE ACCOUNT_NUMBER LIKE '10%'  
4 GROUP BY CUSTOMER_NAME  
5 HAVING COUNT(ACCOUNT_NUMBER)>1  
6 ;  
  
CUSTOMER_NAME      No. Of A/Cs Held  
-----  
Ginny              2  
Ron                2  
Hermione           2  
  
SQL> 
```

Figure 9: Show the Result of Grouping

4. **To perform various types of join.** The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each. Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

5. Implementation of Indexes and Sequences.

6. Implementation of view.

7. Implementation of PL/SQL block.

8. To Perform Various Transaction Control Commands.

9. Implementation of cursor.

(a) **Implicit cursor.**

(b) **Explicit cursor.**

10. Implementation of concurrency control with the help of locking mechanisms.

- (a) Implicit and explicit lock using SQL.**
- (b) Explicit locking using PL/SQL.**
- (c) Implementation of Deadlock.**

11. To perform Error handling in PL/SQL.

12. Implementation of Stored Procedure and Function.

13. Implementation of Oracle Package.

14. Implementation of overloading Procedures and Functions.

15. Implementation of Trigger.

- (a) **Row Trigger**
- (b) **Statement Trigger**
- (c) **Trigger with Exception**