Lab #01

Introduction to Database and Microsoft SQL Server

SQL:

Structured Query Language is a database computer language designed for
managing data in relational database management systems(RDBMS), and originally based upon Relational Algebra. Its scope includes data query and update, schema creation and modification, and data access control. SQL was one of the first languages for Edgar F. Codd's relational model in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks"[3] and became the most widely used language for relational databases.

- IBM developed SQL in mid of 1970's.
- Oracle incorporated in the year 1979.
- SQL used by IBM/DB2 and DS Database Systems.
- SQL adopted as standard language for RDBS by ASNI in 1989.

SQL language is sub-divided into several language elements, including: CLAUSES, which are in some cases optional, constituent components of statements and queries. EXPRESSIONS, which can produce either scalar values or tables consisting of columns and rows of data. PREDICATES which specify conditions that can be evaluated to SQL three-valued logic (3VL) Boolean truth values and which are used to limit the effects of statements and queries, or to change program flow. QUERIES which retrieve data based on specific criteria. STATEMENTS which may have a persistent effect on schemas and data, or which may control transactions, program flow, connections, sessions, or diagnostics. SQL statements also include the SEMICOLON (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.

There are five types of SQL statements.

- Data definition language (DDL)
- Data manipulation language (DML)
- Data retrieval language (DRL)
- Transactional control language (TCL)
- Data control language (DCL)

1. DATA DEFINITION LANGUAGE (DDL): The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structure and usage of four basic DDL commands:

CREATE: This is used to create a new relation and the corresponding ALTER: This is used to add some extra fields into existing relation. DROP TABLE: This is used to delete the structure of a relation. It permanently deletes the records in the table. RENAME: It is used to modify the name of the existing database object. TRUNCATE: This command will remove the data permanently. But structure will not be removed.

Difference between Truncate & Delete:-

- By using truncate command data will be removed permanently & will not get back where as by using delete command data will be removed temporally & get back by using roll back command.
- By using delete command data will be removed based on the condition whereas by using truncate command there is no condition.
- Truncate is a DDL command & delete is a DML command.

2. DATA MANIPULATION LANGUAGE (DML):

The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

INSERT INTO: This is used to add records into a relation. These are three type of INSERT INTO queries which are as UPDATE: This is used to update the content of a record in a relation. DELETE-FROM: This is used to delete all the records of a relation but it will retain the structure of that relation.

3. DATA RETRIEVAL LANGUAGE (DRL):

Retrieves data from one or more tables. SELECT FROM: To display all fields for all records. SELECT - FROM -WHERE: This query is used to display a selected set of fields for a selected set of records of a relation. SELECT - FROM -GROUP BY: This query is used to group to all the records in a relation together for each and every value of a specific key(s) and then display them for a selected set of fields the relation. SELECT - FROM -ORDER BY: This query is used to display a selected set of fields from a relation in an ordered manner base on some field. JOIN using SELECT - FROM - ORDER BY: This query is used to display a set of fields from two relations by matching a common field in them in an ordered manner based on some fields.

JOIN using SELECT - FROM - GROUP BY: This query is used to display a set of fields from two relations by matching a common field in them and also group the corresponding records for each and every value of a specified key(s) while displaying. UNION: This query is used to display the combined rows of two different queries, which are having the same structure, without duplicate rows. INTERSET: This query is used to display the common rows of two different queries, which are having the same structure, and to display a selected set of fields out of them. MINUS: This query is used to display all the rows in relation_1, which are not having in the relation_2.

4. TRANSACTIONAL CONTROL LANGUAGE (TCL): A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be mode permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either role back or commit statement

COMMIT: This command is used to end a transaction only with the help of the commit command transaction changes can be made permanent to the database. SAVE POINT: Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back. Thus, save point is used in conjunction with roll back. ROLE BACK: A role back command is used to undo the current transactions. We can roll back the entire transaction so that all changes made by SQL statements are undo (or) role back a transaction to a save point so that the SQL statements after the save point are roll back.

5. DATA CONTROL LANGUAGE (DCL): DCL provides uses with privilege commands the owner of database objects (tables), has the soul authority ollas them. The owner (data base administrators) can allow other data base uses to access the objects as per their requirement

GRANT: The GRANT command allows granting various privileges to other users and allowing them to perform operations with in their privileges REVOKE: To with draw the privileges that has been GRANTED to a uses, we use the REVOKE command.

TASK:

Visit the below link for the Installation of Microsoft SQL Server 2008.
https://www.youtube.com/watch?v=4WEFTQ3VJNg

https://www.youtube.com/watch?v=WKWZZcrin5I

Lab # 02

Data Definition Operations and to become familiar with data definition language.

DATA DEFINITION LANGUAGE (DDL): The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structure and usage of four basic DDL commands:

CREATE: This is used to create a new table.

Syntax: Create table tablename (column_name1 data_ type constraints, column_name2 data_ type constraints ...) **Example:** Create table dept (deptno NUMBER (2), dname VARCHAR2 (14),loc VARCHAR2(13));

ALTER: This is used to add and modify table. **Example:**

1. Alter table emp add phone_no char (20);

2. Alter table emp alter column phone_ no int;

DROP TABLE: This is used to delete the structure of a relation. It permanently deletes the records in the table. **Example:** drop table prog20; here prog20 is table name

TRUNCATE: This command will remove the data permanently. But structure will not be removed. **Syntax**: TRUNCATE TABLE <TABLE NAME>;

Example: Truncate table customer;

CONSTRAINTS:

In SQL, we have the following constraints:

NOT NULL - Indicates that a column cannot store NULL value
UNIQUE - Ensures that each row for a column must have a unique value
PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have a unique identity which helps to find a particular record in a table more easily and quickly
FOREIGN KEY - Ensure the referential integrity of the data in one table to match values in another table

CHECK - Ensures that the value in a column meets a specific condition
Syntax: Create table <table-name> <column-name**1**> <type**1**> {<column-constraints**1**>], <column- name**2**> <type**2**> {<column-constraints**2**>],.........<column-name**n**> <type**n**> {<column- constraints**n**>].

Example:

Create table emp(empno NUMBER (4), ename VARCHAR2 (10), deptno NUMBER (7,2) NOT NULL, constraint EMP_EMPNO_PK PRIMARY KEY (empno));

Create table emp (empno NUMBER(4), ename VARCHAR2 (10) NOT NULL,deptno NUMBER (7,2) NOT NUL,........., CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno) REFERENCES dept (deptno));

Create table emp (empno NUMBER(4), ename VARCHAR2 (10) NOT NULL,deptno NUMBER (2), CONSTRAINT emp_deptno_ck CHECK (DEPTNO BETWEEN 10 And 99));

TASKS:

- Create a new table **Person** and insert at least 5 records.
- Create a new table **Customer** and insert at least 5 records.
- Create a new table **Order** and insert at least 5 records
- Add a new field in the Person table.
- Modify any field in the Person table.
- Use some constraints in your queries.

Lab #03

Introduction to Data Manipulation Language with Insert ,Update and Delete Command

DATA MANIPULATION LANGUAGE (DML): The Data Manipulation Language (DML) is use to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic

DML commands:

1. INSERT 2. UPDATE 3. DELETE

1. INSERT Only one row is inserted at a line with this command, optionally list the columns in the insert clauses. List values in the default order of the columns in the table.

Encloses character and date values with in single quotation marks.

SYNTAX: Insert into table [(column [, column....])] Values (value [, value...])]

Examples Insert into dept(deptno,dnmae,loc) values (50, 'production', 'san francisco’);

Insert into emp(empno, ename, job, sal, comm, deptno) values (7890,'jinks','clerk',1.2e3,null,40);

2. UPDATE Using update command, we can modify each row in the table

SYNTAX: Update table Set column= value [, column=value] [Where condition]);

If we don’t use the where condition, all the rows in the table will be updated.

Examples Update emp set comm = null where job = ‘CLERK’;

Update emp set job = ‘manager', sal = sal +1000, deptno = 20 where ename=’JONES’;

3. DELETE Removes existing rows by using the delete command.

SYNTAX: Delete [from] table [Where condition];

If the WHERE condition is omitted, all rows in the table will be deleted.

If you will try to delete a row that contains a primary key used as a foreign key in another table, you will experience an integrity constraint error.

Examples Delete from emp where Job = 'SALESMAN';

TASK 1:

- Insert new employee's records.
- Insert new department records deptno=50, dname=ADVERTISING and loc=MIAMI.
- Delete the records of Sales department
- Change the employees sal for Smith
- Confirm all these changes by using Select statement
- Update the salary of each employee to 5000.
- Change the salary to $1,000 for all employees with a salary less than $900.
- Change the commission of department 20 to 1000.
- Change the hire date of all clerks to 02-04-2000
- Delete all the records having hire date before 21-dec-81
- Delete all records where salary is greater than 2000.

TASK2:

- Use insert, update & delete commands in 10 different scenarios.

Lab # 04

Data Retrieval Operations using SELECT, Relationship among Tables considering constraints.

THE SELECT STATEMENT

To select data from a table, the table must be in your own schema or you must have SELECT privilege on the table. To select rows from the base tables of a view, the owner of the schema containing the view must have SELECT privilege on the base tables. Also, if the view is in a schema other than your own, you must have SELECT privilege on the view.

SYNTAX Select < Column Names > From < Table Name >.

EXAMPLES The following statement selects rows from the employee table. Select * from emp; The following statement selects the name, job, salary and department from Emp. Select ename, job, sal, and deptno from emp; The following statement selects all records from Dept. Select * from dept;

COLUMN ALIAS Provides a different name for the column expression and causes the alias to be used in the column heading. The AS keyword is optional. The alias effectively renames the select list item for the duration of the query.

Example: The following Example assigning a temporary column and rename it with Bonus. Select empno, ename, job, sal, sal * 2 “ bonus “ from emp;

Queries with Distinction

If you look at the original table, EMP, you see that some of the data repeats. For Eliminating duplicate rows, use *distinct*

Examples: The following example eliminates duplicate job from EMP.

Select distinct job from EMP;

The following example eliminates duplicate deptno from EMP.

Select distinct deptno from EMP;

CONDITIONS If you ever want to find a particular item or group of items in your database, you need one or more conditions. Conditions are contained in the **WHERE** clause. In the preceding example, the condition is

Syntax Select < Column Names > From < Table Name > Where < Search Condition >.

Select from and where are the three most frequently used clauses in SQL. Where simply causes your queries to be more selective. Without the WHERE clause, the most useful thing you could do with a query is display all records in the selected table(s).

Examples:

The following statement selects rows from the employee table with the department number of 40.

Select * from EMP where deptno = 40;

The following statement selects rows from the employee table with the department number of 40.

Select empno, ename, job, sal from EMP where sal < 3000;

Operators are the elements you use inside an expression to articulate how you want specified conditions to retrieve data.
There are 4 different classes of operators used in SQL, they are

♦ Arithmetic operators ♦ Logical operators ♦ Comparison operators ♦ SQL operators

ARITHMETIC OPERATORS These are the arithmetic operators available in SQL. You may use arithmetic operators in any clause of a SQL statement except the FROM clause.

+ Add

- Subtract

* Multiply

/ Divide

Examples

- Select empno, ename, sal + 2000 from emp;

- Select empno, ename,sal – 100 “deduction” from emp;
- Select empno, ename, sal * comm “New_sal” from emp;

COMPARISON OPERATORS

= Equal to

> Greater than

>= Greater than or equal to

< Less than

<= Less than or equal to

Examples

- Select empno, ename from emp where sal=800;
- Select empno, sal from emp where ename=’Smith’;
- Select * from emp where sal > 1500;
- Select empno, sal, hiredate from emp where sal >=1600;
- Select * from emp where sal < 1500;
- Select * from emp where sal <= 1600;
- Select * from emp where comm*3 < sal;
- Select empno, ename, hiredate from emp where ename =’ALLEN’;
- Select empno, sal, comm from emp where sal! = 1600;
- Select ename, sal, hiredate from emp where sal + comm >=1600;
- Select empno, ename from emp where comm < sal;

SQL OPERATORS There are four SQL operators that operate with all data types:

BETWEEN AND Between two values(inclusive)

IN(list) Match any of a list of values

LIKE Match a character pattern

IS NULL Is a null value

Examples

- Select sal from emp where ename like 'sm';
- Select sal from emp where ename like 'smith';
- Select * from emp where deptno in (10,20);
- Select * from emp where ename in ('smith','allen);
- Select empno, ename from emp where deptno between 10 and 20;
- Select * from emp where sal not between 1000 and 2000;
- Select * from emp where deptno not between 10 and 30;
- Select empno, ename from emp where comm is null;
- Select * from emp where mgr is null;

LOGICAL OPERATOR

AND If both component conditions return TRUE then the result is TRUE

OR If either component condition returns TRUE, then the result is TRUE

NOT Returns the opposite condition

Examples

- Select * from emp where ename='Smith' and deptno=20;
- Select * from emp where job='CLERK' and sal > 800;
- Select * from emp where ename='Smith' or deptno=20;
- Select * from emp where sal<2000 or deptno=20;
- Select * from emp where sal not between 1000 and 1400;
- Select * from emp where deptno not between 10 and 20;

ASSIGNMENTS

1. List all rows of the table emp.
2. List all rows of the dept.
3. List all employees' number from emp.
4. List all employees name from emp.
5. List all departments' number from emp.
6. Find all employees whose salaries are between 500 and 1500.
7. Find all employees whose salaries are between 1500 and 2500.
8. Find all employees whose salaries are between 2600 and 5000.
9. Find all employees whose salaries are less than 2000.
10. Find all employees whose salaries are greater than 2000.

11. Find those employees whose mgr are 7902, 7566, 7788.
12. Find those employees whose mgr is between 7788 and 7092.
13. List all employees whose name starts with ‘s’.
14. List all employees whose name start with 'a'.
15. List all employees having first name as 'scott’.
16. List all employees having first name as ‘smith' or 'king'.
17. List all employees having first letter in their name 'w' or 'k’ and they don’t belong to Department number 10,20 and 40.
18. List all employees whose name begin with 'scott’ and end with martin.
19. List all employees whose name start with 'm' and belong to department number 30.
20. List all employees whose name started with character 'b' and their jobs are manager.
21. List all employees who do not have job manager..
22. Find those employees whose job do not start with ‘a’.
23. Find those employees whose job does not start with 'c'.
24. List of those employees whose mgr is not null.
25. List of those employees whose jobs are 'manager or clerk' of department number 10.
26. List of those employees whose jobs are 'analyst' and 'salesman' of department 30.
27. Find all clerks who earn salaries between 1000 and 2000.
28. Find all managers who earn salaries between 2500 and 3000.
29. Find all employees who are either clerk or 'manager' and all employees who earn Salaries in the range of 1000 and 2000.
30. Find all employees who are either manager and/or all employees who earn salaries between 2000 and 3000.
31. Find all employees whose salaries are equal to 1500 and jobs are manager or salesmen.

LAB # 05

Data Retrieval Operations using Join operations

The JOIN Statement:

One of the most powerful features of SQL is its capability to gather and manipulate data from across several tables. Without this feature you would have to store all the data elements necessary for each application in one table. Without common tables you would need to store the same data in several tables. Imagine having to redesign, rebuild, and repopulate your tables and databases every time your user needed a query with a new piece of information. The JOIN statement of SQL enables you to design smaller, more specific tables that are easier to maintain than larger tables.

SQL JOIN (INNER JOIN) An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them. The most common type of join is:

SQL INNER JOIN (simple join). An SQL INNER JOIN return all rows from multiple tables where the join condition is met.

Syntax SELECT column_name FROM table1 INNER JOIN table2 ON table1.column_name = table2.column_name;

Examples:

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

SELECT Customers.CustomerName, Orders.OrderID FROM Customers INNER JOIN Orders ON Customers.CustomerID=Orders.CustomerID ORDER BY Customers.CustomerName;

SQL LEFT JOIN Keyword : The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL on the right side when there is no match.

Syntax SELECT column_name FROM table1 LEFT JOIN table2 ON table1.column_name = table2.column_name;

Example SELECT Customers.CustomerName, Orders.OrderID FROM Customers LEFT JOIN Orders ON Customers.CustomerID=Orders.CustomerID ORDER BY Customers.CustomerName;

SQL RIGHT JOIN Keyword The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

Syntax SELECT column_name FROM table1 RIGHT JOIN table2 ON table1.column_name = table2.column_name;

Example SELECT Orders.OrderID, Employees.FirstName FROM Orders RIGHT JOIN Employees ON Orders.EmployeeID=Employees.EmployeeID ORDER BY Orders.OrderID;

SQL FULL OUTER JOIN Keyword The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2). The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

Syntax SELECT column_name(s) FROM table1 FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;

Example: SELECT Customers.CustomerName, Orders.OrderID FROM Customers FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID ORDER BY Customers.CustomerName;

TASKS:

(I) Consider the following relation

EMPLOYEE (EMP_ID, EMP_NAME, EMP_ADDRESS, SKILL, PROJ-ID).

EQUIPMENT (EQP-ID, EMP_ID, EQP-TYPE, PROJECT).

- Find the join of relations EMPLOYEE and EQUIPMENT.
- Get all employees for projects using EQP-TYPE as a “Welding machine”.
- Get all machines being used at the Mumbai Project?
- Find all employees of the project using equipment number 110.

(II) Consider the following relation

SALESMAN (SALESMAN_ID, NAME, CITY, COMMISION)

CUSTOMER (CUSTOMER_ID, CUST_NAME, CITY, GRADE, SALESMAN_ID) ORDERS

(ORD_NO, PURCH_AMT, ORD_DATE, CUSTOMER_ID, SALESMAN_ID).

- Write a SQL statement to know which salesman are working for which customer.
- Write a SQL statement to make a list in ascending order for the customer who works either through a salesman or by own.
- Write a SQL statement to make a list in ascending order for the salesmen who works either for one or more customers or not yet joined under any of the customers.
- Write a SQL statement to make a report with customer name, city, order no. order date, purchase amount for those customers from the existing list who placed one or more orders or which order(s) have been placed by the customer who are not in the list.

Lab #06

Single Row Functions & Aggregating Data Using Group Function

ORDER BY CLAUSE

SYNTAX: Select column from table [Where condition] [Order by column] By default rows are sorted by ascending order. By using the Order by clause, we can override this. Two options can be used with the Order_By clause Ascending (Asc, if not mentioned results are sorted in ascending order) Descending (desc, results are sorted, starting with the lowest value)

Examples

- Select ename, job, deptno, and hiredate from emp order by hire date desc;
- Select ename, deptno, and sal from emp order by deptno asc, sal desc;

SINGLE ROW FUNCTIONS

Single row functions are used to manipulate data items; they operate on single rows only, accept one or more arguments and return one result per row. There are basically three types of Single Row Functions. ♦ Character Functions ♦ Number Functions ♦ Conversions.

CHARACTER FUNCTION

LOWER(col): Converts alpha character values to lowercase.
UPPER(col): Converts alpha character values to upper
CONCAT(col1,col2) Concatenates the first character value to the second character value. Equivalent to concatenate operator(||)
SUBSTRING(col m[n]) Returns specified character from character value starting at character position m,n character long. If m is negative, the count starts from the end of the character value.
LEN(col) Returns the number of character in value
CHARINDEX('alphabet' , col): Searches an expression in a string expression and returns its

starting position if found **Examples**

Select lower (ename) from emp;

- Select Upper (ename) from emp;
- select ename,CHARINDEX ('a',ename) from emp;
- select substring(ename,1,2)from emp;

NUMBER FUNCTION

ROUND(column\ expression, n) Rounds the col expression or value to n decimal places. If n is omitted no decimal place. If n is negative ,numbers to the left of the decimal point are rounded
POWER To raise one number to the power of another, use Power. In this function the first argument is raised to the power of the second:
CEILING AND FLOOR Ceil returns the smallest integer greater than or equal to its argument. Floor does just the reverse, returning the largest integer equal to or less than its argument.

CONVERSIONS SYNTAX:

DATEDIFF(datepart,startdate,enddate)
SELECT DATEDIFF(day,'1980-12-17','1982-01-23') AS DiffDate
SELECT GETDATE() AS CurrentDateTime
SELECT DATEPART(yyyy,hiredate) AS OrderYear,
DATEPART(mm,hiredate) AS OrderMonth,
DATEPART(dd,hiredate) AS OrderDay FROM emp

AGGREGATES FUNCTION

COUNT Counts the number of records
SUM Sum of definite column value
AVG Average of specified column value
MAX,MIN To find maximum and minimum values of a column

Examples

- Select COUNT (*) from employees;
- Select SUM (SALARY) from employees;

GROUP BY Aggregate functions are normally used in conjunction with a GROUP BY clause. The GROUP BY clause enables you to use aggregate functions to answer more complex managerial questions such as:

What is the average salary of employees in each department?

How many employees work in each department?

How many employees are working on a particular project?

Group by function establishes data groups based on columns and aggregates the information within a group only. The grouping criterion is defined by the columns specified in GROUP BY clause. Following this hierarchy, data is first organized in the groups and then WHERE clause restricts the rows in each group.

Examples

SELECT deptno , COUNT (*) as total_no FROM emp GROUP BY deptno;

SELECT deptno, job, SUM(Salary) as total_salary FROM emp GROUP BY deptno, Job;

TASKS

1. Group the employees by their salaries.
2. List hiredates in descending order.
3. List hiredates in ascending order
4. List all employees and add 20 rupees in each salary.
5. List all 'manager' and add 100 rupees in each salary.
6. List all 'salesman and add 500 rupees in each salary.
7. List all 'clerk' and add 50 rupees in each salary.
8. Find eight percent of salesmen salary.
9. Find the annual salary of each employee.
10. Find the six-month salary of each employee.
11. Find the two-month salary of each manager.
12. Make a query in which all the arithmetic expressions will include all the result will be remain same.
13. Find the daily wages of each employee.

14. Find the daily wages of each 'manager.
15. Display your name in lowercase.
16. Find the first and second characters of enames.
17. Find the minimum salary from the table emp.
18. Find the maximum salary from the table emp.
19. Find the length of all ename.
20. Find the length of job.
21. Find the sum of all salaries.
22. Find those employees whose department location is Newark.
23. Find those employees who are working in accounting department.
24. Count all employees.
25. Display the sum of all employees' salaries.
26. How many managers do we have?
27. How many departments do we have?
28. List average salary of each job.
29. Find the Maximum and Minimum salary of all employees.
30. Find the average salaries of those employees who work in dept 10.
31. Find average and sum of all the salaries of each job excluding clerks.
32. Find the minimum and average salary of each department excluding deptno 10.

Lab # 07

IMPLEMENTING CARDINALITIES(MINIMUM ,MAXIMUM) AND MAPPING CONSTRAINTS ON MULTIPLE TABLES

Mapping cardinality (cardinality constraints)

It represents the number of entities of another entity set which are connected to an entity using a relationship set.
It is most useful in describing binary relationship sets.
For a binary relationship set the mapping cardinality must be one of the following types:

- One to one
- One to many
- Many to one
- Many to many

TASK :

- Implement mapping of tables from theory classes

Lab #08
SUB-QUERIES IN SQL

SUBQUERIES

Why do we use sub-queries?
Suppose we want to write a query to find out who earns a salary greater than Jones' salary. To solve this problem, we need two queries: one query to find what Jones earns and a second query to find who earns more than that amount. The above problem can be solved by combining the two queries, placing one query inside the other query. The inner query or the sub-query returns a value that is used by the outer query or the main query. Using a sub-query is equivalent to performing two sequential queries and using the result of the first query as the search value in the second query.

Main Query

"Which employees have a salary greater than Jones' Salary?"

Sub-query

"What is Jones' salary?"

Sub-query

A sub-query is a SELECT statement that is embedded in a clause of another SELECT statement. They can be very useful when we need to select rows from a table with a condition that depends on the data in the table itself. The sub-query generally executes first and its output is used to complete the query condition for the main or outer query.

The subquery can be placed in a number of SQL clauses:

WHERE clause, HAVING clause FROM clause

The syntax of SELECT statement using sub-queries is:

SELECT select_list FROM table WHERE expr operator (SELECT select_list FROM table);

Note: In the syntax, operator means comparison operator. Comparison operators fall into two clauses: single-row operators (>, =, >=, <, <>, <=) and multiple-row operators (IN, ANY, ALL).

For example, to display the names of all employees who earn more than employee with empno 7566.

SELECT ename FROM emp WHERE sal > (SELECT sal FROM emp WHERE empno = 7566);

Types of Sub queries

Single-row subquery: Query that returns only one row from the inner SELECT statement.

Multiple-row subquery: Query that returns more than one row form the inner SELECT statement.

Single-Row SubQuery

Examples

1. **To display the employees whose job title is the same as that of employee 7369**

SELECT ename, job FROM emp WHERE job = (SELECT job FROM emp WHERE empno = 7369);

2. To display employees whose job title is the same as that of employee 7369 and whose salary is greater than that of employee 7876.

SELECT ename, job FROM emp WHERE job = (SELECT job FROM emp WHERE empno = 7369) AND sal > (SELECT sal FROM emp WHERE empno = 7876);

3. To display the employee name, job title and salary of all employees whose salary is equal to the minimum salary.

SELECT ename, job, sal FROM emp WHERE sal = (SELECT MIN(sal) FROM emp);

Multiple-Row Subqueries Operator Meaning

IN Equal to any member in the list

ANY Compare value to each value returned by the subquery

ALL Compare value to every value returned by the subquery

Note: The NOT operator can be used with IN, ANY, and ALL operators.

Examples

1. **Find the employees who earn the same salary as the minimum salary for departments**.

SELECT ename, sal, deptno FROM emp WHERE sal IN (SELECT MIN(sal) FROM emp GROUP BY deptno);

2. To display employees whose salary is less than any clerk and who are not clerks.

SELECT empno, ename, job FROM emp WHERE sal < ANY (SELECT sal FROM emp WHERE job = ‘CLERK’);

3. To display employees whose salary is greater than the average salary of all the departments.

SELECT empno, ename, job FROM emp WHERE sal > ALL (SELECT avg(sal) FROM emp GROUP BY deptno);

Multiple-Column Sub queries

If we want to compare two or more columns, we must write a compound WHERE clause using logical operators. Multiple column subqueries enable us to combine duplicate WHERE conditions into a single WHERE clause.

For example, to display the name of all employees who have done their present

job somewhere before in their career.

SELECT ENAME FROM EMP WHERE (EMPNO, JOB) IN (SELECT EMPNO, JOB FROM JOB_HISTORY);

COMPOUND QUERIES

In SQL, we can use the normal set operators of Union, Intersection and Set Difference to combine the results of two or more component queries into a single result table. Queries containing SET operators are called *compound* queries. The following table shows the different set operators provided in Oracle SQL.

Operator	Returns
UNION	All distinct rows selected by either query
UNION ALL	All rows selected by either query including all duplicates
INTERSECT	All distinct rows selected by both queries
MINUS	All distinct rows that are selected by the first SELECT statement and that are not selected in the second SELECT statement

There are restrictions on the tables that can be combined using the set operations, the most important one being that the two tables have to be union-compatible; that is, they have the same structure. This implies that the two tables must contain the same number of columns, and that their corresponding columns contain the same data types and lengths. It is the user's responsibility to ensure that values in corresponding columns come from the same domain. For example, it would not be sensible to combine a column containing the age of staff with the number of rooms in a property, even though both columns may have the same data type i.e. NUMBER.

The UNION Operator

The UNION operator returns rows from both queries after eliminating duplicates. By default, the output is sorted in ascending order of the first column of the SELECT clause.

For example to display all the jobs that each employee has performed, the following query will be given. (NOTE: If an employee has performed a job multiple times, it will be shown only once)

SELECT EMPNO, JOB FROM JOB_HISTORY UNION

SELECT EMPNO, JOB FROM EMP;

The UNION ALL Operator

The UNION ALL operator returns rows from both queries including all duplicates. For example to display the current and previous jobs of all employees, the following query will be given. (NOTE: If an employee has performed a job multiple times, it will be shown separately).

SELECT EMPNO, JOB FROM JOB_HISTORY UNION ALL

SELECT EMPNO, JOB FROM EMP;

The INTERSECT Operator

The INTERSECT operator returns all rows that are common to both queries. For example, to display all employees and their jobs those have already performed their present job somewhere else in the past.

SELECT EMPNO, JOB FROM JOB_HISTORY INTERSECT

SELECT EMPNO, JOB FROM EMP;

The MINUS Operator

The MINUS operator returns rows from the first query that is not present in the second query. For example to display the ID of those employees whose present job is the first one in their career.

SELECT EMPNO, JOB FROM EMP

MINUS

SELECT EMPNO, JOB FROM JOB_HISTORY;

TASK:

1. Write a query to display the employee name and hiredate for all employees in the same department as Blake. Exclude Blake.
2. Create a query to display the employee number and name for all employees who earn more than the average salary.
3. Write a query to display the employee number and name for all employees who work in a department with any employee whose name contains a T.
4. Display the employee name, department number, and job title for all employees whose department location is Dallas.
5. Display the employee name and salary of all employees who report to King.
6. Write a query to display the employee name, salary, deptno and job for all employees in the same job as empno 7369.
7. Display the employee number, name and salary for all employees who earn more than the average salary and who work in department with any employee with a T in their name.

Lab #09

Retrieving Data from Multiple Tables, Mapping of shared data SQL:

Cartesian product.

A Cartesian Product results when all rows in the first table are joined to all rows in the second table. A Cartesian product is formed under following conditions:-
i. When a join condition is omitted
ii. When a join condition is invalid

Consider the following example:-

SELECT * FROM EMP, DEPT;

In the above example, if EMP table has 14 rows and DEPT table has 4 rows, then their Cartesian product would generate 14 x 4 = 56 rows.
In fact, the ISO standard provides a special format of the SELECT statement for the Cartesian product:-

SELECT * FROM EMP CROSS JOIN DEPT;

A Cartesian product tends to generate a large number of rows and its result is rarely useful. It is always necessary to include a valid join condition in a WHERE clause. Hence a join is always a subset of a Cartesian product.

Types of Joins

There are various forms of join operation, each with subtle differences, some more useful than others. The Oracle 9i database offers join syntax that is SQL 1999 compliant. Prior to release 9i, the join syntax was different from the ANSI standard. However, the new syntax does not offer any performance benefits over the Oracle proprietary join syntax that existed in prior releases.

Inner-Join/Equi-Join

If the join contains an equality condition, it is called equi-join.

Examples

To retrieve the employee name, their job and department name, we need to extract data from two tables, EMP and DEPT. This type of join is called *equijoin*-that is, values in the DEPTNO column on both tables must be equal. Equijoin is also called *simple join* or *inner join*.

```
SELECT E.ENAME, E.JOB, D.DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO;
```

The SQL-1999 standard provides the following alternative ways to specify this join:-

```
SELECT ENAME, JOB, DNAME
FROM EMP NATURAL JOIN DEPT;
```

Outer-Join

A join between two tables that returns the results of the inner join as well as unmatched rows in the left or right tables is a left or right outer join respectively. A full outer join is a join between two tables that returns the results of a left and right join.

Left Outer Join

```
SELECT E.ENAME, D.DEPTNO, D.DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO(+);
```

NOTE: The outer join operator appears on only that side that has information missing.

The SQL-1999 standard provides the following alternative way to specify this join:-

```
SELECT E.ENAME, D.DEPTNO, D.DNAME
FROM EMP E LEFT OUTER JOIN DEPT D
ON (E.DEPTNO = D.DEPTNO);
```

Right Outer Join

```
SELECT E.ENAME, D.DEPTNO, D.DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO(+) = D.DEPTNO;
```

The SQL-1999 standard provides the following alternative way to specify this join:-

```
SELECT E.ENAME, D.DEPTNO, D.DNAME
FROM EMP E RIGHT OUTER JOIN DEPT D
ON (E.DEPTNO = D.DEPTNO);
```

NOTE: In the equi-join condition of EMP and DEPT tables, department OPERATIONS does not appear because no one works in that department. In the outer join condition, the OPERATIONS department also appears.

Full Outer Join

The SQL-1999 standard provides the following way to specify this join:-

```
SELECT E.ENAME, D.DEPTNO, D.DNAME
FROM EMP E FULL OUTER JOIN DEPT D
ON (E.DEPTNO = D.DEPTNO);
```

Non-Equijoin

If the join contains inequality condition, it is called non-equijoin. E.g. to retrieve employee name, salary and their grades using *non-equijoins*, we need to extract data from two tables, EMP and SALGRADE.

```
SELECT E.ENAME, E.SAL, S.GRADE
FROM EMP E, SALGRADE S
WHERE E.SAL
BETWEEN S.LOSAL AND S.HISAL;
```

Self Join

To find the name of each employee's manager, we need to join the EMP table to itself, or perform a *self join*.

```
SELECT WORKER.ENAME || ' works for ' ||
MANAGER.ENAME FROM EMP WORKER, EMP
MANAGER
WHERE WORKER.MGR = MANAGER.EMPNO;
```

TASK:

- To display the employee name, department name, and location of all employees who earn a commission.
- To display all the employee's name (including KING who has no manager) and their manager name
- To display the name of all employees whose manager is ***KING***.
- Create a unique listing of all jobs that in department 30. Include the location of department 30 in the Output.
- Write a query to display the name, job, department number and department name for all employees who work in New York
- Display the employee name and employee number along with their manager's name Manager Number. Label the columns Employee, Emp#, Manager, and Manager#, respectively.

LAB # 10

CREATING SEQUENCES AND INDEXES

SEQUENCE

Sequence is a set of integers 1, 2, 3, … that are generated and supported by some database systems to produce unique values on demand.

- A sequence is a user defined schema bound object that generates a sequence of numeric values.
- Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.
- The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when exceeds max_value.
- One can select the NEXTVAL and CURRVAL from a sequence. Selecting the NEXTVAL will automatically increment the sequence.

Syntax:

```
CREATE SEQUENCE sequence_name

START WITH initial_value

INCREMENT BY increment_value

MINVALUE minimum value

MAXVALUE maximum value

CYCLE| NOCYCLE ;
```

sequence_name: Name of the sequence.
initial_value: starting value from where the sequence starts.
Initial_value should be greater than or equal
to minimum value and less than equal to maximum value. **increment_value:**
Value by which sequence will increment itself. Increment_value can be positive or negative.
minimum_value: Minimum value of the sequence.
maximum_value: Maximum value of the sequence.
cycle: When sequence reaches its set_limit
it starts from the beginning.
nocycle: An exception will be thrown if
the sequence exceeds its max_value.

Example:

```
CREATE  SEQUENCE  sequence_1
start with 1
increment by 1
minvalue 0
maxvalue   100
cycle;

CREATE  SEQUENCE  sequence_2
start with 100
increment by -1
minvalue 1
maxvalue   100
cycle;
```

Example to use sequence

create a table named students with columns as id and name.

```
CREATE  TABLE  students(
ID int,
NAME char(20)
);
```

Now insert values into table

```
INSERT into students VALUES (next value for sequence1,'ali');
INSERT into students VALUES (next value for sequence1,'salman');
```

Output:

```
ID     NAME
----------------------------------------
  1    ali
  2    salman
```

Sequence vs. Identity columns

Sequences, different from the identity columns, are not associated with a table. The relationship between the sequence and the table is controlled by applications. In addition, a sequence can be shared across multiple tables.

The following table illustrates the main differences between sequences and identity columns:

Property/Feature	**Identity**	**Sequence Object**
Allow specifying minimum and/or maximum increment values	No	Yes
Allow resetting the increment value	No	Yes
Allow caching increment value generating	No	Yes
Allow specifying starting increment value	Yes	Yes
Allow specifying increment value	Yes	Yes
Allow using in multiple tables	No	Yes

When to use sequences

You use a sequence object instead of an identity column in the following cases:

- The application requires a number before inserting values into the table.
- The application requires sharing a sequence of numbers across multiple tables or multiple columns within the same table.
- The application requires to restart the number when a specified value is reached.
- The application requires multiple numbers to be assigned at the same time. Note that you can call the stored procedure sp_sequence_get_range to retrieve several numbers in a sequence at once.
- The application needs to change the specification of the sequence like maximum value.

INDEXES

An index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically. An index provides direct and fast access to rows in a table. Its purpose is to reduce the necessity of disk I/O by using an indexed path to locate data quickly. The index is used and maintained automatically by the Oracle Server. Once an index is created, no direct activity is required by the user.

Indexes are logically and physically independent of the table they index. Therefore, they can be created or dropped at any time and have no effect on the base tables or other indexes.

Types of indexes

Oracle maintains the indexes automatically: when new rows are added to the table, updated, or deleted, Oracle updates the corresponding indexes. We can create the following indexes:-

Bitmap index

A bitmap index does not repeatedly store the index column values. Each value is treated as a key, and for the corresponding ROWIDs a bit is set. Bitmap indexes are suitable for columns with low cardinality, such as the GENDER column in the EMP table, where the possible values are M or F. The cardinality is the number of distinct column values in a column. In the EMP table column, the cardinality of the GENDER column is 2.

B-tree index

This is the default. The index is created using the b-tree algorithm. The b-tree includes nodes with the index column values and the ROWID of the row. The ROWIDs are used to identify the rows in the table.
The following are the types of b-tree indexes:-

- Unique Index: The Oracle server automatically creates this index when a column in a table is defined to be a PRIMARY KEY or UNIQUE key contraint.
- NonUnique Index: Users can create nonunique indexes on columns to speed up access time to the rows. For example, we can create a FOREIGN KEY column index for a join in a query to improve retrieval speed.
- Function-based index: The function-based index can be created on columns with expressions. For example, creating an index on the SUBSTR(EMPID, 1, 2) can speed up the queries using the SUBSTR(EMPID, 1, 2) in the WHERE clause.

Creating an Index

To create an index (b-tree) on ENAME column in the EMP table.

```
CREATE INDEX emp_ename_idx
ON emp(ename);
```

To create an index (b-tree) on first 5 characters of JOB column in the EMP table.

CREATE INDEX emp_job5_idx
ON emp(SUBSTR(JOB, 1, 5));

To create a bitmap index, we must specify the keyword BITMAP immediately after CREATE. Bitmap indexes cannot be unique. For example:

CREATE BITMAP INDEX IND_PROJ_STAT ON PROJECT (STATUS);

Confirming Indexes

We can confirm the existence of indexes from the USER_INDEXES data dictionary view. It contains the name of the index and its uniqueness.
SELECT INDEX_NAME, TABLE_NAME, TABLE_OWNER, UNIQUENESS FROM USER_INDEXES;

Removing an Index

It is not possible to modify an index. To change it, we must drop it first and then re-create it. Remove an index definition from the data dictionary by issuing the DROP INDEX statement. To drop an index, one must be the owner of the index or have the DROP ANY INDEX privilege.

DROP INDEX *index*;

For example, remove the EMP_ENAME_IDX index from the data dictionary.
DROP INDEX emp_ename_idx;

When to create an index

The index should be created under following circumstances:-

- The column is used frequenctly in the WHERE clause or in a join condition.
- The column contains a wide range of values.
- The column contains a large number of null values
- Two or more columns are frequently used together in a WHERE clause or a join condition.
- The table is large and most queries are expected to retrieve less than 2-4% of the rows.

When not to create an index
The index should not be created under following circumstances:-

- The table is small
- The columns are not often used as a condition in the query.
- Most queries are expected to retrieve more than 2-4% of the rows.
- The table is updated frequently.

TASKS:

1. CREATE SEQUENCE my_sequence MINVALUE 1 MAXVALUE 1000 START WITH 1 INCREMENT BY 2;
2. Create a new table **Person** with my_sequence and insert at least 5 records.
3. Create a new table **Customer** with my_sequence and insert at least 5 records
4. Create **B-Tree** indexes on i) **Name** column of EMP table ii) **Designation** column of EMP table iii) First 10 characters of **Title** in TRAINING table
5. Create **bitmapped** indexes on i) **Gender** column of EMP table ii) **Performance** column of EMP_PROJECT table

Lab # 11

CREATING TABLES & VIEWS

Tables can be created at any time, even while users are using the database.
We do not need to specify the size of any table. The size is ultimately defined by the amount of space allocated to the database as a whole.
Table structure can be modified online.

- **Table**: Stores data
- **View**: Subset of data from one or more tables
- **Sequence**: Generates primary key values
- **Index**: Improves the performance of some queries

Naming Conventions

- Name database tables and columns according to the standard rules for naming any database object.
- Table names and column names must begin with a letter and can be 1-30 characters long.
- Names must contain only the characters A-Z, a-z, 0-9, _*(underscore*), $, and # (legal characters, but their use is discouraged).
- Names must not duplicate the name of another object owned by the same Server user.
- Names must not be a Server reserved word.

Creating and Altering Tables

The CREATE TABLE statement

To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create objects. The database administrator uses data control language (DCL) statements, covered in a later session, to grant privileges to users
The syntax is as follows:-
CREATE TABLE [*schema* .] *table*
(*column datatype* [DEFAULT *expr*] [, …]);

Referencing another user's tables

A *schema* is a collection of objects. *Schema objects* are the logical structures that directly refer to the data in a database. Schema objects include tables, views, synonyms, sequences, stored procedures, indexes, clusters, and database links.
If a table does not belong to the user, the owner's name must be prefixed to the table.

The DEFAULT option

A column can be given a default value by using the DEFAULT option. This option prevents null values from entering the columns if a row is inserted without a value for the column. The default value can be a literal, an expression, or a SQL function, such as SYSDATE and USER, but the value cannot be the name of another column or a pseudocolumn, such as NEXTVAL, or CURRVAL. The default expression must match the datatype of the column.

For example,
… hiredate DATE DEFAULT SYSDATE, …

Example

The following example creates the DEPT table mentioned in the lab. session 01.

```
CREATE TABLE dept
(deptno NUMBER(2), dname
VARCHAR2(14),            loc
                VARCHAR2(13)
);
```

Since creating a table is a DDL statement, an automatic commit takes place when this statement is executed.

SQL Data Types

Datatype	Description
VARCHAR2(*size*)	Variable-length character data (A maximum *size* must be specified. Default and minimum *size* is 1; maximum *size* is 4000)
CHAR(*size*)	Fixed-length character data of length *size* bytes (Default and minimum *size* is 1; maximum *size* is 2000)
NUMBER(*p*, *s*)	Number having precision p and scale s (The precision is the total number of decimal digits and the scale is the number of digits to the right of the decimal point. The precision can range from 1 to 38 and the scale can range from -84 to 127.)
DATE	Date and time values between January 1, 4712 B.C. and December 31, 9999 A.D.
LONG	Variable length character data up to 2 gigabytes

Datatype	Description
CLOB	Single-byte character data up to 2 gigabytes
RAW(*size*)	Raw binary data of length *size* (A maximum size must be specified. Maximum *size* is 2000.)
LONG RAW	Raw binary data of variable length up to 2 gigabytes
BLOB	Binary data up to 4 gigabytes
BFILE	Binary data stored in an external file; up to 4 gigabytes

CLOB, BLOB and BFILE are the large object data types and can store blocks of unstructured data (such as text, graphics images, video clips and sound wave forms up to 4 gigabytes in size.) LOBs also support random access to data.

Creating a table by using a Subquery

The following example creates a table, DEPT30, that contains details of all employees working in department 30

```
CREATE TABLE  dept30
AS SELECT empno, ename, sal * 12 ANNSAL, hiredate
FROM  emp
WHERE deptno = 30;
```

Altering table structure

The ALTER TABLE statement is used to

- Add a new column
- Modify an existing column
- Define a default value for the new column

The following example adds a new column to the DEPT30 table:-

```
ALTER TABLE dept30
ADD (job VARCHAR2(9));
```

To modify an existing column, use

```
ALTER TABLE dept30
MODIFY          (ename  VARCHAR2(15));
```

Dropping a Table

The DROP TABLE statement removes the definition of an Oracle table. The database loses all the data in the table and all the indexes associated with it.

The DROP TABLE statement, once executed, is irreversible. The Oracle Server does not question the action when the statement is issued and the table is immediately dropped. All DDL statements issue a commit, therefore, making the transaction permanent.

To drop the table DEPT30,

```
DROP TABLE DEPT30;
```

Changing the name of an object

To change the name of a table, view, sequence, or synonym, execute the RENAME statement:-

```
RENAME dept TO department;
```

What are constraints?

The Oracle Server uses constraints to prevent invalid data entry into tables.
Constraints are used for the following purposes:-

- Enforce rules at the table level whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the deletion of a table if there are dependencies from other tables.

Constraint	**Description**
NOT NULL	Specifies that this column may not contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a foreign key relationship between the column and a column of the referenced table
CHECK	Specifies a condition that must be true

The EMP table is being created specifying various constraints:-

```
CREATE TABLE DEPT (
DEPTNONUMBER(2) constraint DEPT_DEPTNO_PK PRIMARY KEY,
DNAME VARCHAR2(14),
LOC VARCHAR2(13),
CONSTRAINT DEPT_DNAME_UK UNIQUE(DNAME));

CREATE TABLE EMP (
EMPNO NUMBER(4) CONSTRAINT EMP_EMPNO_PK PRIMARY KEY,
ENAME VARCHAR2(10) NOT NULL,
JOB VARCHAR2(9),
MGR NUMBER(4),
HIREDATE DATE DEFAULT SYSDATE,
SAL NUMBER(7, 2),
COMM NUMBER(7, 2),
DEPTNO NUMBER(7, 2) NOT NULL,
CONSTRAINT EMP_DEPTNO_CK CHECK (DEPTNO BETWEEN 1 AND 50),
CONSTRAINT EMP_DEPTNO_FK FOREIGN KEY (DEPTNO)
REFERENCESDEPT(DEPTNO));
```

Composite primary keys are defined at the table leve

SQL CREATE VIEW STATEMENT:

In SQL, a view is a virtual table based on the result-set of an SQL statement.
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

What is a View?
A view is a logical table based on other tables or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables in which a view is based are called *base tables*. The view is stored as a SELECT statement in the data dictionary.

Why Use Views?
Views are used for following reasons:-

- To restrict database access because the view can display a selective portion of the database.
- To make complex queries easy. For example, views allow users to query information from multiple tables without knowing how to write a join statement.
- To allow data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- To present different views of the same data to different users.

Simple and Complex Views
There are two classifications for views: simple and complex. The basic difference is related to the DML (insert, update and delete) operations.

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML through view	Yes	Not always

CREATE VIEW SYNTAX:

```
CREATE VIEW view_name AS
SELECT column1, column2,...
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

CREATE VIEW EXAMPLE 1:

The following SQL creates a view that shows all customers from Brazil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName , ContactName
FROM Customers
WHERE Country = 'Brazil';
```

We can query the view above as follows:

```
SELECT * FROM [Brazil Customers];
```

CREATE VIEW EXAMPLE 2:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG (Price) FROM Products);
```

We can query the view above as follows:

```
SELECT * FROM [Products Above Average Price];
```

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW command.

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2,...
FROM table_name
WHERE condition;
```

The following SQL adds the "City" column to the "Brazil Customers" view:

```
CREATE OR REPLACE VIEW [BrazilCustomers] AS
SELECT CustomerName,ContactName,City
FROM Customers
WHERE Country = 'Brazil';
```

SQL Dropping a View

A view is deleted with the DROP VIEW command.

SQL DROP VIEW Syntax

DROP VIEW *view_name*;

The following SQL drops the "Brazil Customers" view:

```
DROP VIEW [Brazil Customers];
```

TASKS

- Consider the following schema, in the form of normalized relations, to represent information about *employees*, *grades*, *training* and *projects* in an organization.

EMPLOYEE
Empno (eg 6712) Name
Designation (e.g. *Database Developer*) Qualification
Joindate PROJECT PID (eg P812)
Title Client
Duration (in weeks)
Status (New, In Progress, Complete) EMP_PROJECT
Empno PID
Performance (Excellent, Good, Fair, Bad, Poor)

GRADE
Designation Grade (1-20) TotalPosts
PostsAvailable (<= TotalPosts)
TRAINING
Tcode (eg T902) Title
StartDate EndDate
EMP_TRAINING

Empno Tcode
Attendance (%)

1. Develop a script file EMPLOYEE.SQL to create tables for the above schema. Implement all necessary integrity constraints including primary and foreign keys. (NOTE: All check constraints should be at table level)

2. Write SQL statements to add
    - Gender column to EMP table. The only possible values are Male and Female.
    - Instructor_Name column to TRAINING table.
    - Salary column to GRADE table.

3. Write down a transaction to insert data in EMP_TRAINING table. The data should be finally saved in the database.
    - Employee 3400 gets Developer 6i training and his attendance is 87%
    - Employee 3300 gets Typing/shorthand training and her attendance

Lab # 12

Managing profiles and controlling user access

Controlling database access and resource limits are important aspects of the DBA's function. Profiles are used to manage the database and system resources and to manage database passwords and password verification. Database access is controlled using privileges.

Roles are created to manage the privileges.

PROFILES

In Oracle, a profile is a named set of password and resource limits. In essence, they are used for two major purposes:-

i. Profiles are used to control the *database* and *system* resource usage. They restrict users from performing some operations that require heavy use of resources. Resource management limits can be enforced at the session level, the call level or both. Oracle provides a set of predefined *resource parameters* that can be used to monitor and control database resource usage. The DBA can define limits for each resource by using a database *profile.*

ii. Profiles are also used for *password management* to provide greater control over database security. e.g. to force a user to change password after a specified time.

It is possible to create various profiles for different user communities and assign a profile to each user. When the database is created, Oracle creates a profile named DEFAULT, and if we do not specify a profile for the user, the DEFAULT profile is assigned.

Oracle lets us control the following types of resource usage (both at session level and call level) through profiles:-

- Concurrent sessions per user
- Elapsed and idle time connected to the database
- CPU time used (per session and per call)
- Logical reads performed (per session and per call)

Oracle lets us control the following password management features through profiles:-

- Password aging and expiration
- Password history
- Password complexity verification
- Account Locking

Enforcing resource limits

Resource limits are enforced only when the parameter RESOURCE_LIMIT is set to TRUE. Even if the profiles are defined and assigned to users, Oracle enforces them only when this parameter is set to TRUE. This parameter can be set in the initialization parameter file so that every time the database starts, the resource usage is controlled for each user using the assigned profile. Resource limits can also be enabled or disabled using the ALTER SYSTEM command. The default value of RESOURCE_LIMIT is FALSE.

```
ALTER SYSTEM
SET RESOURCE_LIMIT = TRUE;
```

Most resource limits are set at the session level; a session is created when a user connects to the database. Certain limits can be controlled at the statement level (but not at the transaction level).

If a user exceeds a resource limit, Oracle aborts the current operation, rolls back the changes made by the statement, and returns an error. The user has the option of committing or rolling back the transaction, because the statements issued earlier in the transaction are not aborted. No other operation is permitted when a session level limit is reached. The user can disconnect, in which case the transaction is committed. In contrast to resource limits, password limits are always enforced.

Some of the parameters that are used to control *resources* are as follows:-

- SESSIONS_PER_USER
- CPU_PER_SESSION
- CPU_PER_CALL
- LOGICAL_READS_PER_SESSION
- LOGICAL_READS_PER_CALL
- CONNECT_TIME
- IDLE_TIME

Some of the parameters that are used for *password* management are as follows:-

- FAILED_LOGIN_ATTEMPTS
- PASSWORD_LOCK_TIME
- PASSWORD_LIFE_TIME
- PASSWORD_GRACE_TIME
- PASSWORD_REUSE_TIME
- PASSWORD_REUSE_MAX

Creating Profiles

Let's create a profile to manage passwords and resources for the accounting department users. The users are required to change their password every **60** days and they cannot reuse a given password for **90** days. However a given password can be reused any number of times throughout life. The grace period for password change is 5 days. They are allowed to make a **typo** in the password only **four** consecutive times while connecting to the database; if the login fails for a fifth time, their account will be locked forever (until the DBA or security department unlocks the account). The accounting department users are allowed to have a maximum of **six** database connections; they can stay connected to the database for **24** hours, but an inactivity of half-an-hour will terminate their session.

```
CREATE PROFILE ACCOUNTING_USER LIMIT
SESSIONS_PER_USER                6
CONNECT_TIME                     1440

IDLE_TIME                    30
FAILED_LOGIN_ATTEMPTS        4
PASSWORD_LOCK_TIME           UNLIMITED
PASSWORD_LIFE_TIME           60
PASSWORD_GRACE_TIME          5
PASSWORD_REUSE_TIME          90
PASSWORD_REUSE_MAX           UNLIMITED;
```

Dropping profiles

Profiles are dropped using the DROP PROFILE command. If any user is assigned the profile you wish to drop, Oracle returns an error. You can drop such profiles by specifying CASCADE, in which case the users who have that profile will be assigned the DEFAULT profile.

```
DROP PROFILE ACCOUNTING_USER CASCADE;
```

Assigning Profiles

Profiles can be assigned to users by using the CREATE USER or ALTER USER command. Following example assigns the ACCOUNTING_USER profile to an existing user named SCOTT:-

```
ALTER USER SCOTT
PROFILE ACCOUNTING_USER;
```

Querying Password and Resource Limits Information

Information about password and resource limits can be obtained by querying the following data dictionary views:-

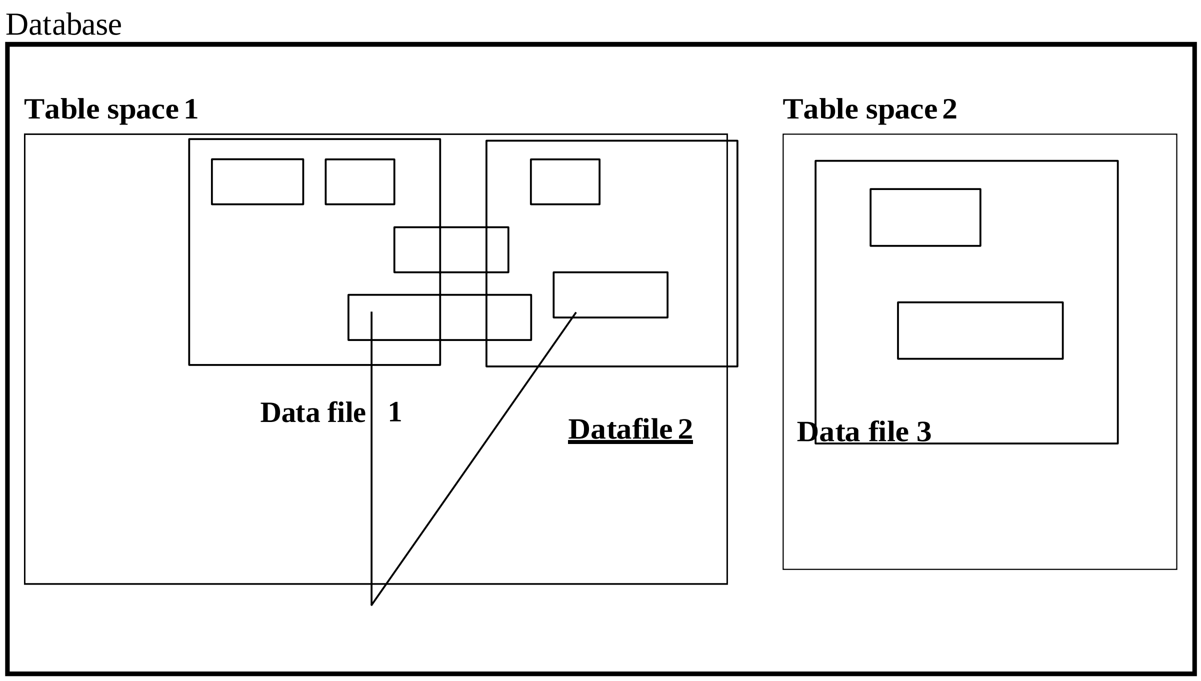
- DBA_USERS
- DBA_PROFILES

TABLESPACES AND DATAFILES

The database's data is stored logically in *tablespaces* and physically in the *data files* corresponding to the tablespaces. The logical storage management is independent of the physical storage of the data files. A tablespace can have more than one data file associated with it whereas one data file belongs to only one tablespace.
A database can have one or more tablespaces.
Figure 10.2 below shows the relationship between the database, tablespace, data files and

the objects in the database.

Any object (such as a table, an index, etc.) created in the database is stored in a single table space. But the object's physical storage can be on multiple data files belonging to that table space.
The idea of table spaces allows managing space for users in an efficient way. We can group application-related data (e.g. accounts) together so that when maintenance is required for the application's table space, only that table space need to be taken offline and the rest of the database will be available for other users. Moreover, we can back up the database one table space at a time as well as make part of the database read-only.
The size of the table space is the total size of all the data files belonging to the table space. The size of the database is the total size of all table spaces in the database, which is the total size of all data files in the database.
Changing the size of the data files belonging to a table space can change the size of that table space. More space can be added to a table space by adding more data files to the table space or increasing the size of the existing data files.



A Database can have multiple table spaces and a table space can have multiple data files

Objects

When a database is created, Oracle creates the SYSTEM table space. All data dictionary objects are stored in it. The data files specified at the time of database creation are assigned to the SYSTEM table space. The PL/SQL program units (such as procedures, functions, packages, or triggers) created in the database are also stored in the SYSTEM table space.
It is recommended that SYSTEM table space should contain only the objects in the data dictionary. Separating the data dictionary from other database objects reduces the contention between dictionary objects and database objects for the same data file.

Creating Table space

The following statement creates a table space ACCOUNT_DATA having two data files acct_data01.dbf and acct_data02.dbf,

```
CREATE TABLESPACE ACCOUNT_DATA
DATAFILE            'd:\oradata\db01\acct_data01.dbf'      SIZE
100M AUTOEXTEND ON NEXT 5M MAXSIZE 200M,
'd:\oradata\db01\acct_data02.dbf' SIZE 100M;
```

AUTOEXTEND ON enables the automatic extension of the data file. NEXT specifies the disk space to allocate to the data file when more space is needed. MAXSIZE specifies the maximum disk space allowed for allocation to the data file. If MAXSIZE is specified unlimited then this indicates that there is no limit on allocating disk space to the data file.

Altering Table space

We can add more space to a table space by adding more data files to it or by changing the size of the existing data files.

To add more data files,

```
ALTER TABLESPACE ACCOUNT_DATA ADD DATAFILE
'd:\oradata\db01\acct_data03.dbf' SIZE 100M;
```

We can increase or decrease the size of a data file by using the RESIZE clause of the ALTER DATABASE DATAFILE command.

```
ALTER DATABASE
DATAFILE      'd:\oradata\db01\appl_data01.dbf'
RESIZE 200M;
```

Dropping Table space

The following statement drops the table space APPL_DATA,
DROP TABLESPACE APPL_DATA;

Undo Table space

An undo table space is used to store undo segments. It cannot contain any other objects. Unlike other table spaces, the undo table space is limited to the DATAFILE. For example, to create an undo table space undo1
CREATE UNDO TABLESPACE undo1
DATAFILE ‘d:\oradata\db01\undo01.dbf’ SIZE 40M;

Querying Table space Information

Information about tablespaces can be obtained by querying the following data dictionary views:-

- DBA_TABLESPACES
- V$TABLESPACE

MANAGING USERS

The following statement will create a user AHMED identified by password GREEN, having profile ACCOUNTING_USER. All the user’s objects will be created in the USERS table space in which the user AHMED will have a quota of 2MB. The user’s account will be initially locked.

CREATE USER AHMED
IDENTIFIED BY GREEN DEFAULT
TABLESPACE USERS QUOTA 2M
ON USERS
PROFILE ACCOUNTING_USER
ACCOUNT LOCK;

The DBA can later lock or unlock a user’s account as follows:-
ALTER USER AHMED
ACOUNT UNLOCK;

The DBA can also expire the user’s password:
ALTER USER AHMED PASSWORD EXPIRE;
Users must change the password the next time they log in, or the DBA must change the password. If the password is expired, SQL*Plus prompts for a new password at login time.

The following example shows how to drop the user AHMED, with all the owned objects:-
DROP USER AHMED CASCADE;

CONTROLLING USER ACCESS

PRIVILEGES

Privileges in the Oracle database control access to the data and restrict the actions users can perform. Through proper privileges, users can create, drop, or modify objects in their own schema or in another user's schema. Privileges also determine what data a user should have access to.

Privileges can be granted to a user via two methods:

- Assign privileges directly to the user
- Assign privileges to a role, and then assign the role to the user
-

A *role* is a named set of privileges, which eases the management of privileges. For example, if there are 10 users needing access to the data in the accounting tables, we can grant the privileges required to a role and grant the role to the 10 users.

There are two types of privileges:

Object privileges are granted on a specific object. The owner of the object has all the privileges on the object. The privileges can be on data (to read, modify, delete, add, or reference), on a program (to execute), or to modify an object (to change the structure).
System privileges are the privileges that enable the user to perform an action on any schema in the database. They do not specify an object, but are granted at the database level. Like object privileges, system privileges also can be granted to a user. They are usually granted by DBA. Both system privileges and object privileges can be granted to a role.

PUBLIC is a user group defined in the database; it is not a database user or a role. Every user in the database belongs to this group. So if privileges are granted to PUBLIC, they are available to all users of the database.

Granting Object Privileges

Suppose user AHMED owns tables CUSTOMER (Customer_ID, Name, Address) and ORDER (Order_ID, Date, Customer_ID). AHMED wants to grant read and update privileges on CUSTOMER table to user YASIR. When multiple privileges are specified, they are separated by comma.

```
GRANT SELECT, UPDATE ON CUSTOMER
TO YASIR;
```

The INSERT and UPDATE privileges can be granted on columns also.
GRANT INSERT(Customer_id, Name), DELETE ON CUSTOMER
TO YASIR **WITH GRANT OPTION**;

The WITH GRANT OPTION clause allows YASIR to grant the privileges to others.

Granting System Privileges

Like object privileges, system privileges also can be granted to a user, role, or PUBLIC. There are many system privileges in Oracle. The CREATE, ALTER, and DROP privileges provide the ability to create, modify, and drop the object specified in the user's schema. When a privilege is specified with ANY, it authorizes the user to perform the action on any schema in the database.
For example, to grant create session, create table and create index privilege to AHMED:- GRANT CREATE SESSION, CREATE TABLE,

CREATE INDEX TO AHMED **WITH ADMIN OPTION**;
The WITH ADMIN OPTION clause allows AHMED to grant the privileges to others.

The following statement allows AHMED to create and select a table in any schema as well as selecting any sequence,
GRANT CREATE ANY TABLE, SELECT ANY
TABLE, SELECT ANY SEQUENCE TO AHMED;

Revoking privileges

Object privileges and system privileges can be revoked from a user by using the REVOKE statement. To revoke the UPDATE privilege granted to YASIR from AHMED on AHMED's CUSTOMER table:-

REVOKE UPDATE
ON CUSTOMER
FROM YASIR;

ROLES

A role is a named group of related privileges that can be granted to the user. This method makes it easier to revoke and maintain privileges.
A user can have access to several roles and several users can be assigned the same role. Roles are typically created for a database application.

e.g. to create a role *manager*
CREATE ROLE manager

To grant privileges to the manager role

```
GRANT CREATE TABLE, CREATE VIEW
TO manager;
```

To grant role to users,

```
GRANT manager TO AHMED, YASIR;
```

Lab # 13
SQL Auditing

Introduction:

There are several levels of auditing for SQL Server, depending on government or standards there are tree several types of Database audits but tree distinct types are mentioned below.

A. Creating a server audit with a file target
B. Creating a server audit with a Windows Application log target with options
C. Creating a server audit containing a WHERE clause

SQL Server Audit provides the tools and processes you must have to enable, store, and view audits on various server and database objects. You can record server audit action group's per-instance, and either database audit action groups or database audit actions per database. The audit event will occur every time that the auditable action is encountered.

SQL Server Database Engine or an individual database involves tracking and logging events that occur on the Database Engine. SQL Server audit lets you create server audits, which can contain server audit specifications for server level events, and database audit specifications for database level events. Audited events can be written to the event logs or to audit files.

Syntax

```
CREATESERVERAUDITaudit_name
{
TO{ [FILE (<file_options> [ , ...n ] ) ] | APPLICATION_LOG | SECURITY_LOG | URL |
EXTERNAL_MONITOR }
[ WITH ( <audit_options> [ , ...n ] ) ]
[ WHERE<predicate_expression> ]
}
[ ; ]

<file_options>::=
{
FILEPATH = 'os_file_path'
[ ,MAXSIZE = { max_size { MB | GB | TB } | UNLIMITED } ]
[ , { MAX_ROLLOVER_FILES = { integer | UNLIMITED } } | { MAX_FILES = integer } ]
[ ,RESERVE_DISK_SPACE = { ON | OFF } ]
}

<audit_options>::=
{
   [   QUEUE_DELAY = integer ]
[ ,ON_FAILURE = { CONTINUE | SHUTDOWN | FAIL_OPERATION } ]
[ ,AUDIT_GUID = uniqueidentifier ]
}

<predicate_expression>::=
{
   [NOT ]<predicate_factor>
[ {AND | OR } [NOT ] { <predicate_factor> } ]
   [,...n ]
}
```

A. Creating a server audit with a file target

The following example creates a server audit called HIPAA_Audit with a binary file as the target and no options.

```
CREATESERVERAUDITHIPAA_Audit
TOFILE( FILEPATH ='\\SQLPROD_1\Audit\' );
```

B. Creating a server audit with a Windows Application log target with options

The following example creates a server audit called HIPAA_Audit with the target set for the Windows Application log. The queue is written every second and shuts down the SQL Server engine on failure.

```
CREATESERVERAUDITHIPAA_Audit
TO APPLICATION_LOG
WITH( QUEUE_DELAY = 1000,  ON_FAILURE = SHUTDOWN);
```

C. Creating a server audit containing a WHERE clause

The following example creates a database, schema, and two tables for the example. The table named DataSchema.SensitiveData contains confidential data and access to the table must be recorded in the audit. The table named DataSchema.GeneralData does not contain confidential data. The database audit specification audits access to all objects in the DataSchema schema. The server audit is created with a WHERE clause that limits the server audit to only the SensitiveData table. The server audit presumes an audit folder exists at C:\SQLAudit.

```
CREATEDATABASETestDB;
GO
USETestDB;
GO
CREATESCHEMADataSchema;
GO
CREATETABLEDataSchema.GeneralData (IDint PRIMARY KEY, DataFieldvarchar(50)
NOTNULL);
GO
CREATETABLEDataSchema.SensitiveData (IDint PRIMARY KEY, DataFieldvarchar(50)
NOTNULL);
GO
-- Create the server audit in the master database
USEmaster;
GO
CREATESERVERAUDITAuditDataAccess
```

```
TOFILE( FILEPATH ='C:\SQLAudit\' )
   WHERE object_name = 'SensitiveData' ;
GO
ALTER SERVER AUDIT AuditDataAccess WITH (STATE = ON);
GO
-- Create the database audit specification in the TestDB database
USE TestDB;
GO
CREATE DATABASE AUDIT SPECIFICATION [FilterForSensitiveData]
FOR SERVER AUDIT [AuditDataAccess]
ADD (SELECT ON SCHEMA::[DataSchema] BY [public])
WITH (STATE = ON);
GO
-- Trigger the audit event by selecting from tables
SELECT ID, DataField FROM DataSchema.GeneralData;
SELECT ID, DataField FROM DataSchema.SensitiveData;
GO
-- Check the audit for the filtered content
SELECT * FROM fn_get_audit_file('C:\SQLAudit\AuditDataAccess_*.sqlaudit',default,default);
GO
```

Task.

1. Creating a server audit with a file target
2. Creating a server audit with a Windows Application log target with options
3. Creating a server audit containing a WHERE clause

Lab #14

Apriori Algoritm (association rule Mining)

What Is an Itemset

A set of items together is called an item set. If any item set has k-items it is called a k-item set. An item set consists of two or more items. An item set that occurs frequently is called a frequent item set. Thus frequent item set mining is a data mining technique to identify the items that often occur together.If an item set is frequent, then all of its subsets must also be frequent. Conversely, if a subset is infrequent, then all of its supersets must be infrequent too.

What Is a Frequent Item set.

A set of items is called frequent if it satisfies a minimum threshold value for support and confidence. Support shows transactions with items purchased together in a single transaction. Confidence shows transactions where the items are purchased one after the other.

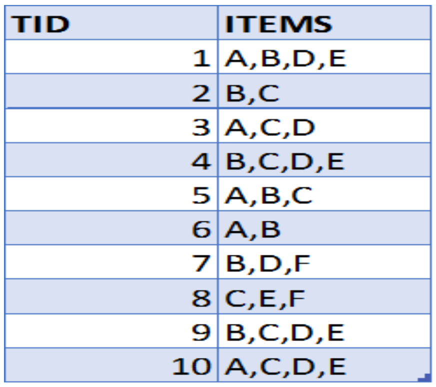
Frequent Pattern Mining (FPM)

The frequent pattern mining algorithm is one of the most important techniques of data mining to discover relationships between different items in a dataset. These relationships are represented in the form of association rules. It helps to find the irregularities in data.

THEORY:

Consider the following dataset and we will find frequent itemsets and generate association rules for them.

Data Set:



TID	ITEMS
1	A,B,D,E
2	B,C
3	A,C,D
4	B,C,D,E
5	A,B,C
6	A,B
7	B,D,F
8	C,E,F
9	B,C,D,E
10	A,C,D,E

First, created a table named "APR_ITEMS_BOUGHT" with the following script and manually inserted the data into the table:

```
Create tale APR_items_bought
(TID nmber ,Items varchar2(50));
```

After creating the table above, the item sets are inserted in columns to a new table. The script for generating the table is:

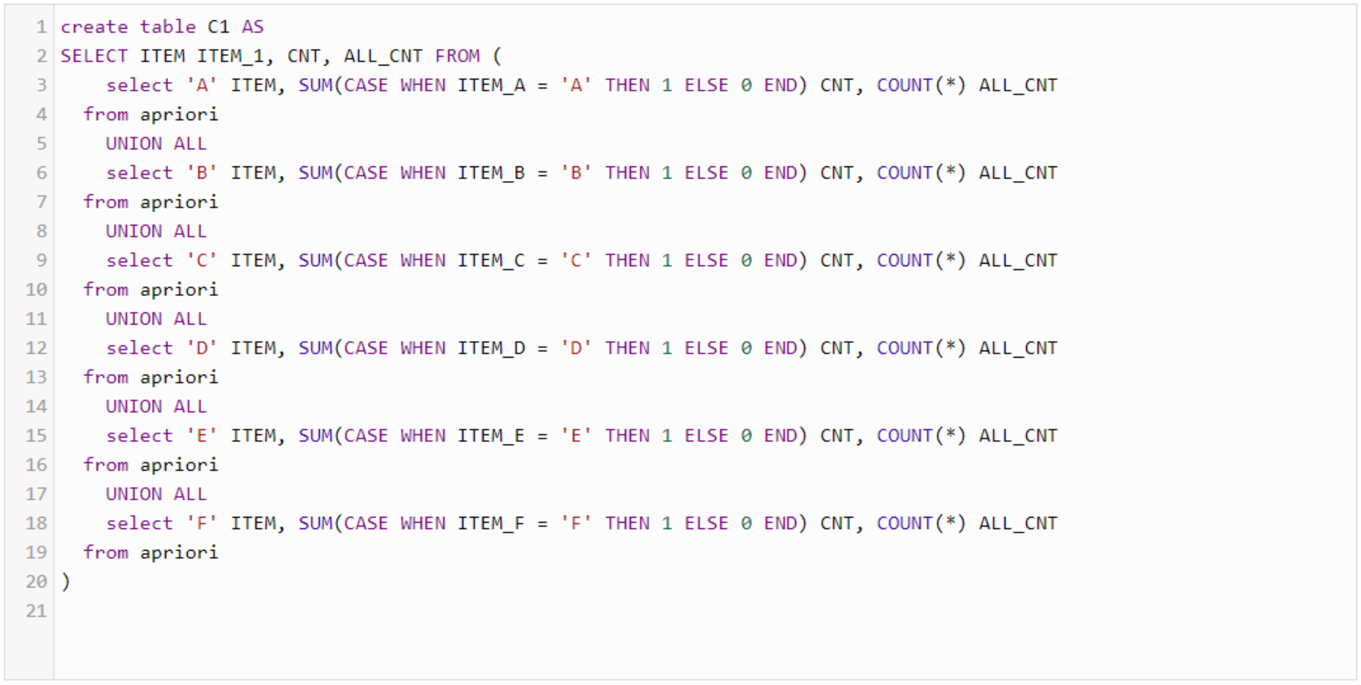
```
Create tabeleApriori(TIDnumber,Item_Avarchar2(10),Item_B varchar2(10),
```

Item_Cvarchar2 (10),Item_Dvarchar2 (10),Item_Evarchar2 (10), Item_Fvarchar2 (10));

Finding the frequent item sets has two steps:

- Generating candidate items set
- Generating frequent items set

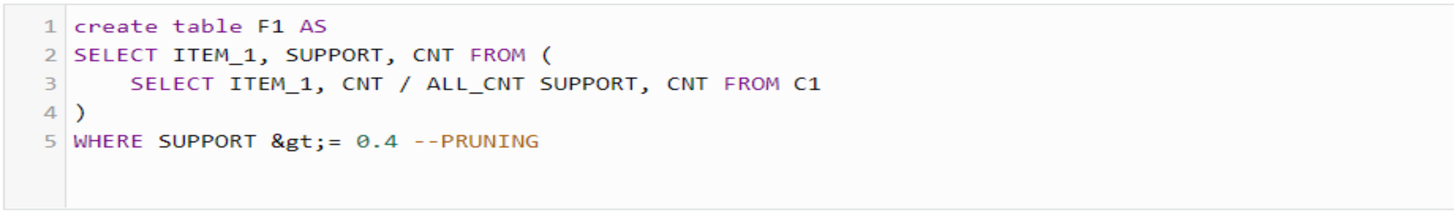
Now the item set generation can be started. The first candidate table "C1" is created in the following script;



```
create table C1 AS
SELECT ITEM ITEM_1, CNT, ALL_CNT FROM (
    select 'A' ITEM, SUM(CASE WHEN ITEM_A = 'A' THEN 1 ELSE 0 END) CNT, COUNT(*) ALL_CNT
  from apriori
    UNION ALL
    select 'B' ITEM, SUM(CASE WHEN ITEM_B = 'B' THEN 1 ELSE 0 END) CNT, COUNT(*) ALL_CNT
  from apriori
    UNION ALL
    select 'C' ITEM, SUM(CASE WHEN ITEM_C = 'C' THEN 1 ELSE 0 END) CNT, COUNT(*) ALL_CNT
  from apriori
    UNION ALL
    select 'D' ITEM, SUM(CASE WHEN ITEM_D = 'D' THEN 1 ELSE 0 END) CNT, COUNT(*) ALL_CNT
  from apriori
    UNION ALL
    select 'E' ITEM, SUM(CASE WHEN ITEM_E = 'E' THEN 1 ELSE 0 END) CNT, COUNT(*) ALL_CNT
  from apriori
    UNION ALL
    select 'F' ITEM, SUM(CASE WHEN ITEM_F = 'F' THEN 1 ELSE 0 END) CNT, COUNT(*) ALL_CNT
  from apriori
)
```

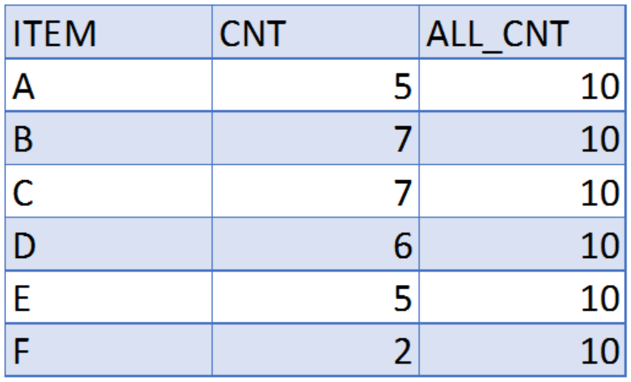
Then the frequent itemset is created. I choose min support and min confidence values as:

- Min support %40
- Min confidence %60
-



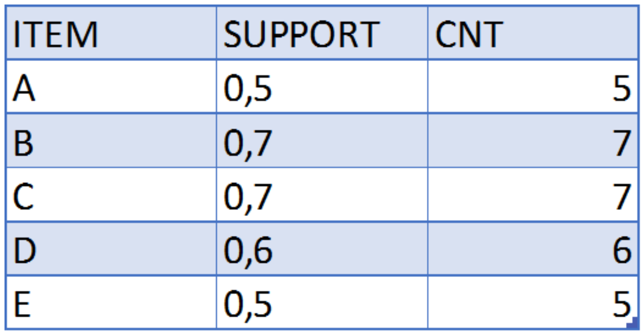
```
create table F1 AS
SELECT ITEM_1, SUPPORT, CNT FROM (
    SELECT ITEM_1, CNT / ALL_CNT SUPPORT, CNT FROM C1
)
WHERE SUPPORT &gt;= 0.4 --PRUNING
```

The result of both the "C1" and "F1" sets are shown below;



ITEM	CNT	ALL_CNT
A	5	10
B	7	10
C	7	10
D	6	10
E	5	10
F	2	10

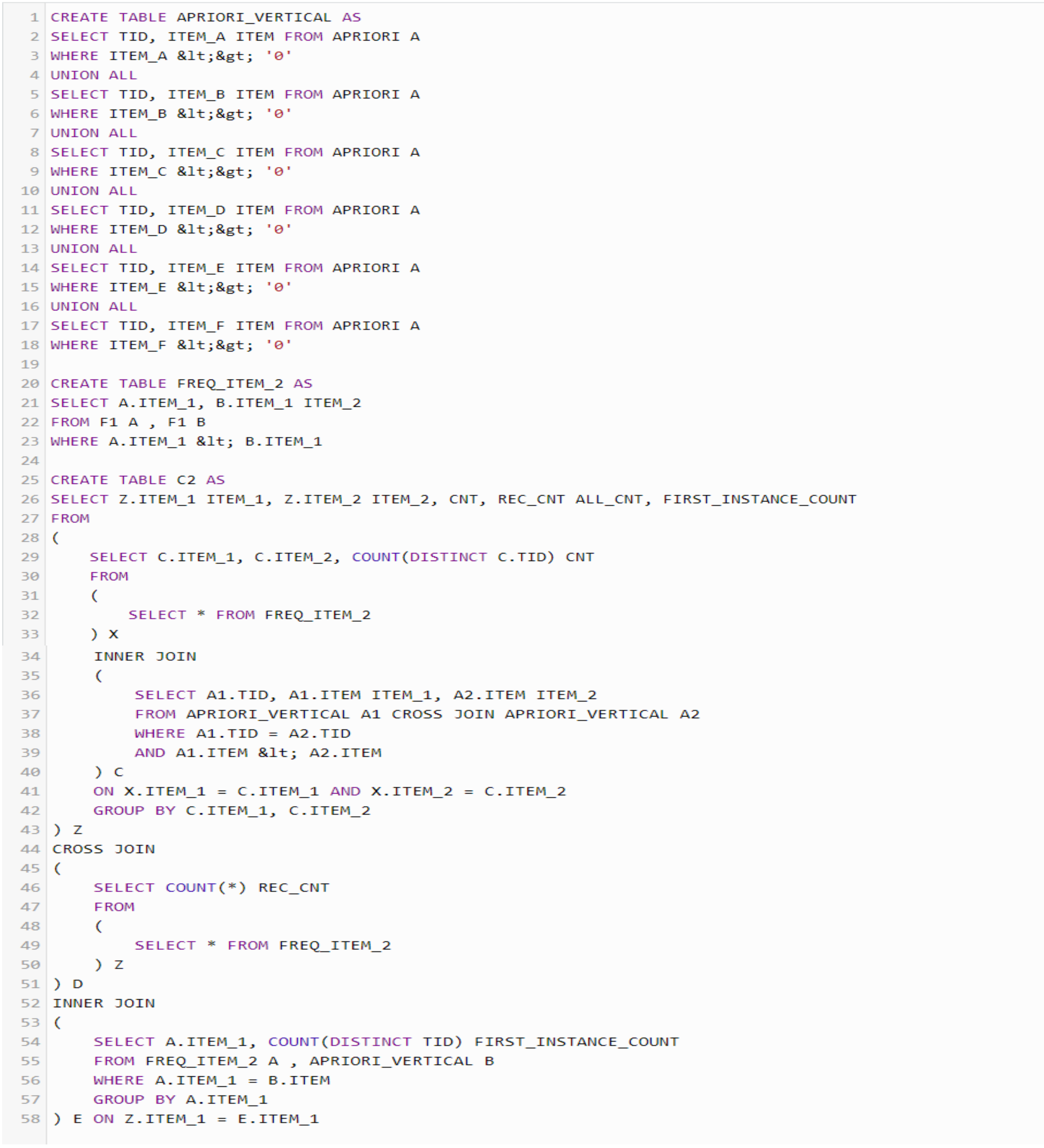
The C1 table has three columns: ITEM column gives the items in the market basket, CNT column gives how many transactions the item has, and ALL_CNT column gives all transactions count. After pruning with a %40 support value, we are left with the following frequent item sets:



ITEM	SUPPORT	CNT
A	0,5	5
B	0,7	7
C	0,7	7
D	0,6	6
E	0,5	5

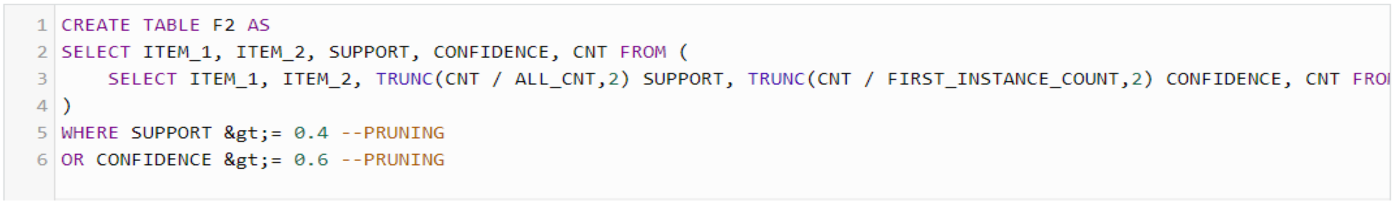
The F1 table has also three columns: the ITEM column, SUPPORT value for each column, and the transaction count in the CNT column.

For the second item set generation and frequent item set, the pruned F1 table will be used.



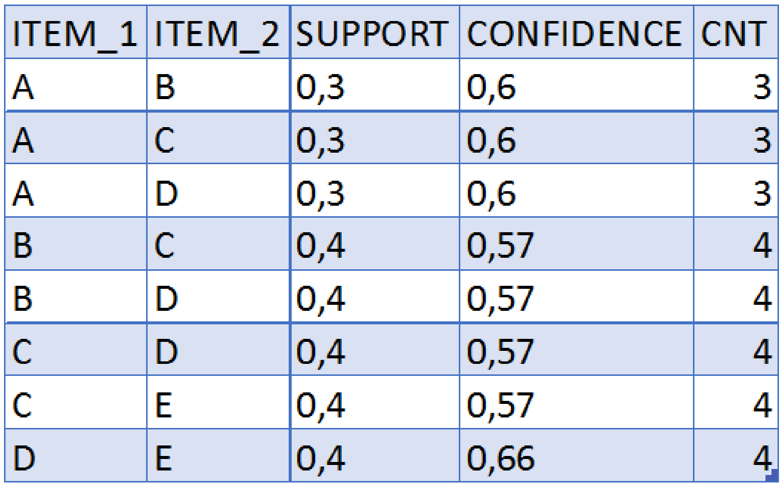
```
CREATE TABLE APRIORI_VERTICAL AS
SELECT TID, ITEM_A ITEM FROM APRIORI A
WHERE ITEM_A &lt;&gt; '0'
UNION ALL
SELECT TID, ITEM_B ITEM FROM APRIORI A
WHERE ITEM_B &lt;&gt; '0'
UNION ALL
SELECT TID, ITEM_C ITEM FROM APRIORI A
WHERE ITEM_C &lt;&gt; '0'
UNION ALL
SELECT TID, ITEM_D ITEM FROM APRIORI A
WHERE ITEM_D &lt;&gt; '0'
UNION ALL
SELECT TID, ITEM_E ITEM FROM APRIORI A
WHERE ITEM_E &lt;&gt; '0'
UNION ALL
SELECT TID, ITEM_F ITEM FROM APRIORI A
WHERE ITEM_F &lt;&gt; '0'

CREATE TABLE FREQ_ITEM_2 AS
SELECT A.ITEM_1, B.ITEM_1 ITEM_2
FROM F1 A , F1 B
WHERE A.ITEM_1 &lt; B.ITEM_1

CREATE TABLE C2 AS
SELECT Z.ITEM_1 ITEM_1, Z.ITEM_2 ITEM_2, CNT, REC_CNT ALL_CNT, FIRST_INSTANCE_COUNT
FROM
(
    SELECT C.ITEM_1, C.ITEM_2, COUNT(DISTINCT C.TID) CNT
    FROM
    (
        SELECT * FROM FREQ_ITEM_2
    ) X
    INNER JOIN
    (
        SELECT A1.TID, A1.ITEM ITEM_1, A2.ITEM ITEM_2
        FROM APRIORI_VERTICAL A1 CROSS JOIN APRIORI_VERTICAL A2
        WHERE A1.TID = A2.TID
        AND A1.ITEM &lt; A2.ITEM
    ) C
    ON X.ITEM_1 = C.ITEM_1 AND X.ITEM_2 = C.ITEM_2
    GROUP BY C.ITEM_1, C.ITEM_2
) Z
CROSS JOIN
(
    SELECT COUNT(*) REC_CNT
    FROM
    (
        SELECT * FROM FREQ_ITEM_2
    ) Z
) D
INNER JOIN
(
    SELECT A.ITEM_1, COUNT(DISTINCT TID) FIRST_INSTANCE_COUNT
    FROM FREQ_ITEM_2 A , APRIORI_VERTICAL B
    WHERE A.ITEM_1 = B.ITEM
    GROUP BY A.ITEM_1
) E ON Z.ITEM_1 = E.ITEM_1
```

F2 table



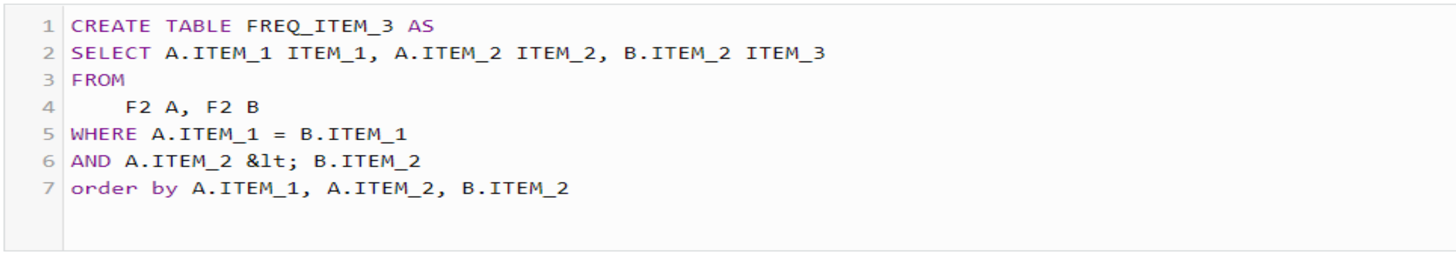
```
CREATE TABLE F2 AS
SELECT ITEM_1, ITEM_2, SUPPORT, CONFIDENCE, CNT FROM (
    SELECT ITEM_1, ITEM_2, TRUNC(CNT / ALL_CNT,2) SUPPORT, TRUNC(CNT / FIRST_INSTANCE_COUNT,2) CONFIDENCE, CNT FRO
)
WHERE SUPPORT &gt;= 0.4 --PRUNING
OR CONFIDENCE &gt;= 0.6 --PRUNING
```

After creating the F2 table, we are left with the following records, which means the most frequent couples:



ITEM_1	ITEM_2	SUPPORT	CONFIDENCE	CNT
A	B	0,3	0,6	3
A	C	0,3	0,6	3
A	D	0,3	0,6	3
B	C	0,4	0,57	4
B	D	0,4	0,57	4
C	D	0,4	0,57	4
C	E	0,4	0,57	4
D	E	0,4	0,66	4

The most frequent items for the third step:



```
CREATE TABLE FREQ_ITEM_3 AS
SELECT A.ITEM_1 ITEM_1, A.ITEM_2 ITEM_2, B.ITEM_2 ITEM_3
FROM
    F2 A, F2 B
WHERE A.ITEM_1 = B.ITEM_1
AND A.ITEM_2 &lt; B.ITEM_2
order by A.ITEM_1, A.ITEM_2, B.ITEM_2
```

The 3rd C3 table:

```
create table C3 AS
SELECT Z.ITEM_1 ITEM_1, Z.ITEM_2 ITEM_2, Z.ITEM_3 ITEM_3, CNT, REC_CNT ALL_CNT, FIRST_INSTANCE_COUNT
FROM
(
    SELECT C.Item_1, C.ITEM_2, C.ITEM_3, COUNT(DISTINCT C.TID) CNT
    FROM
    (
        SELECT * FROM FREQ_ITEM_3
    ) X
    INNER JOIN
    (
        SELECT A1.TID, a1.item ITEM_1, a2.item ITEM_2, A3.ITEM ITEM_3
        FROM APRIORI_VERTICAL A1
        CROSS JOIN APRIORI_VERTICAL A2
        CROSS JOIN APRIORI_VERTICAL A3
        where A1.TID = A2.TID
        AND A1.TID = A3.TID
        AND A1.ITEM &lt; A2.ITEM
        AND A1.ITEM &lt; A3.ITEM
        AND A2.ITEM &lt; A3.ITEM
    ) C
    ON X.ITEM_1 = C.ITEM_1 AND X.ITEM_2 = C.ITEM_2 AND X.ITEM_3 = C.ITEM_3
    GROUP BY C.ITEM_1, C.ITEM_2, C.ITEM_3
) Z
CROSS JOIN
(
    select COUNT(*) REC_CNT
    FROM
    (
        SELECT * FROM FREQ_ITEM_3
    ) Z
) D
INNER JOIN
(
    SELECT A.ITEM_1, A.ITEM_2, COUNT(DISTINCT B.TID) FIRST_INSTANCE_COUNT
    FROM FREQ_ITEM_3 A , APRIORI_VERTICAL B, APRIORI_VERTICAL C
    WHERE A.ITEM_1 = B.ITEM
    AND A.ITEM_2 = C.ITEM
    AND B.TID = C.TID
    GROUP BY A.ITEM_1, A.ITEM_2
) E ON Z.ITEM_1 = E.ITEM_1 AND Z.ITEM_2 = E.ITEM_2

```

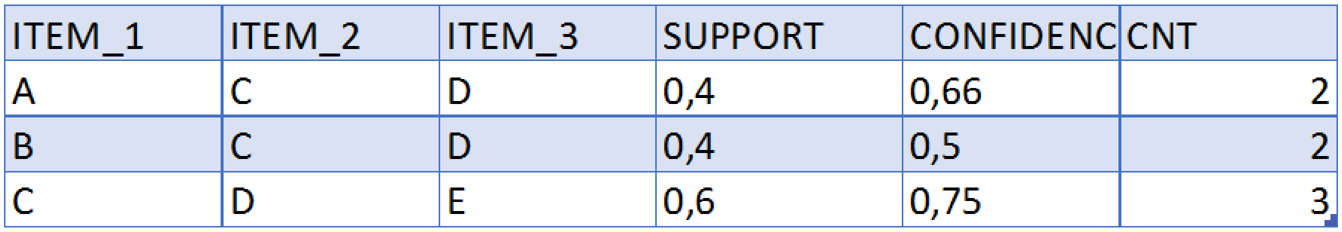
The 3rd pruning step for frequent itemsets, the F1 table:

```
create table F3 AS
SELECT ITEM_1, ITEM_2, ITEM_3, SUPPORT, CONFIDENCE, CNT FROM (
   SELECT ITEM_1, ITEM_2, ITEM_3, TRUNC(CNT / ALL_CNT,2) SUPPORT, TRUNC(CNT / FIRST_INSTANCE_COUNT,2) CONFIDENCE,
)
WHERE SUPPORT &gt;= 0.4 --PRUNING
OR CONFIDENCE &gt;= 0.6 --PRUNING
```

In the 4th step, there is no data in the frequent itemset table:

```
CREATE TABLE FREQ_ITEM_4 AS
SELECT A.ITEM_1 I1, A.ITEM_2 I2, a.ITEM_3, b.item_3 ITEM_4
FROM
    F3 A
    , F3 B
WHERE A.ITEM_1 = B.ITEM_1
AND A.ITEM_2 = B.ITEM_2
AND A.ITEM_3 &lt; B.ITEM_3
order by A.ITEM_1, A.ITEM_2, a.ITEM_3, b.item_3
```

So we can conclude that the most frequent itemsets are:



ITEM_1	ITEM_2	ITEM_3	SUPPORT	CONFIDENC	CNT
A	C	D	0,4	0,66	2
B	C	D	0,4	0,5	2
C	D	E	0,6	0,75	3

Lab Task.

1-Create and construct your own dataset by considering 65%confidence and 50 % support and also conclude/output of the most frequent set in SQL.