UNIT 8: SECURITY AND EXPLAIN

- DB2   SECURITY   - STATIC SQL
- DB2   SECURITY   - DYNAMIC SQL
- HOW TO INVOKE EXPLAIN.
- COMMIT AND ROLLBACK.

Figure: 8.1 Security

GRANT AND REVOKE

SECURITY definitions are made by issuing GRANT statements, such as

```
GRANT SELECT ON TABLE EMP TO userid
```

The opposite, withdrawing a GRANTed privileges, is done with the REVOKE Statement :

```
REVOKE SELECT ON TABLE EMP FROM userid
```

DB2 SECURITY - STATIC SQL

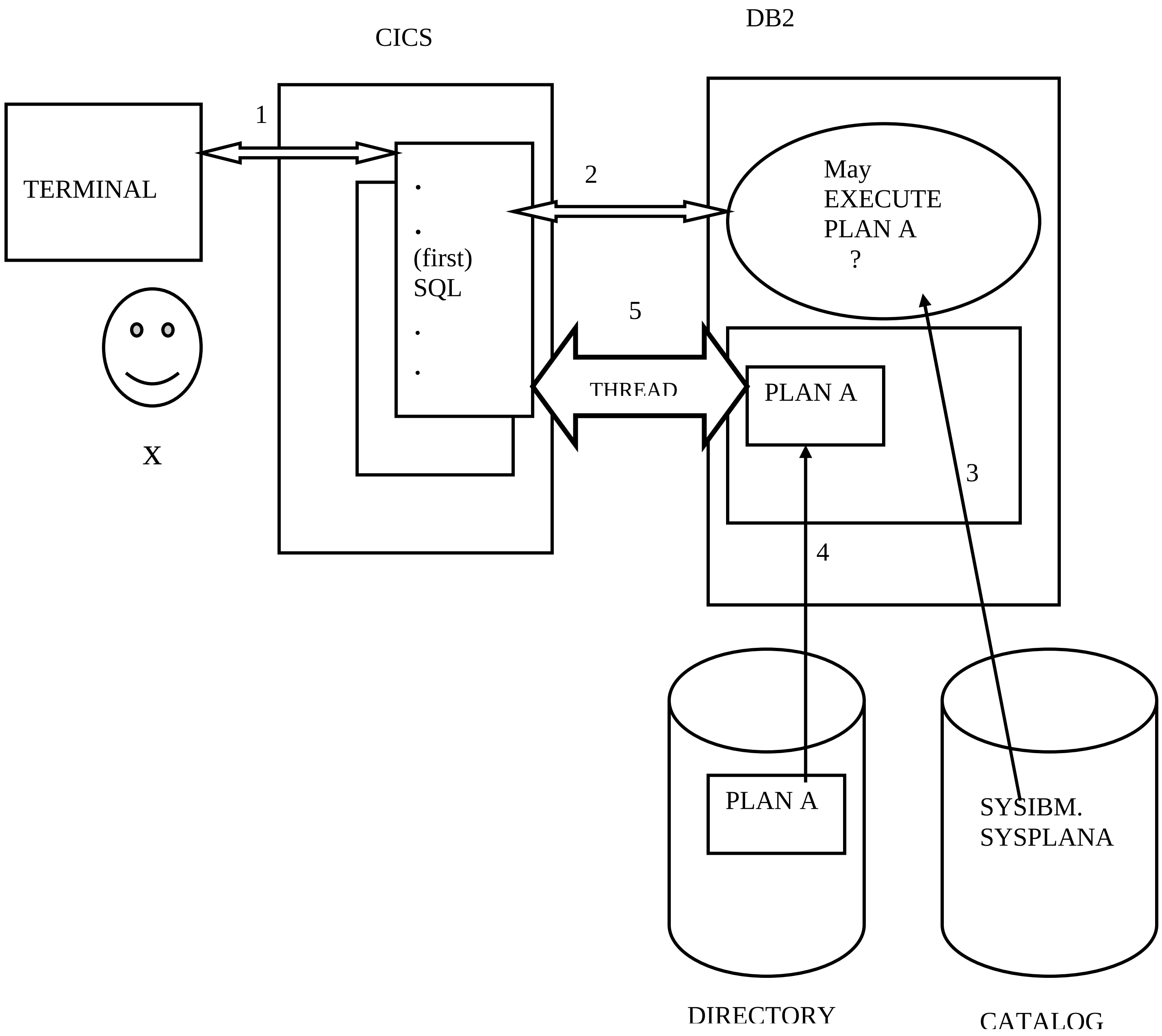


Figure 9.1 DB2 Security – Static SQL

DB2 SECURITY - STATIC SQL(Contd…)

Let’s take the example of a CICS transaction program containing SQL calls.

At BIND PACKAGE time, security checks were done to make sure that the BINDer could create the package. When the program was put into production, the EXECUTE privilege on the plan was GRANTed to the users that need to execute it.

What happens at execution time ?

1. User “x” enters a transaction code on his terminal(transid) leading to the execution of load module LMODA in CICS
2. The application invokes the first SQL statement.
3. At this first SQL statement, DB2 checks whether “x” has the EXECUTE privilege on PLANA.
4. If so, it will load the plan into storage and execute it. The plan will invoke the necessary packages.
5. The “thread” is created. From now on, no more security checks will be made. Subsequent SQL calls will be execute over the thread.

DB2 SECURITY - DYNAMIC SQL

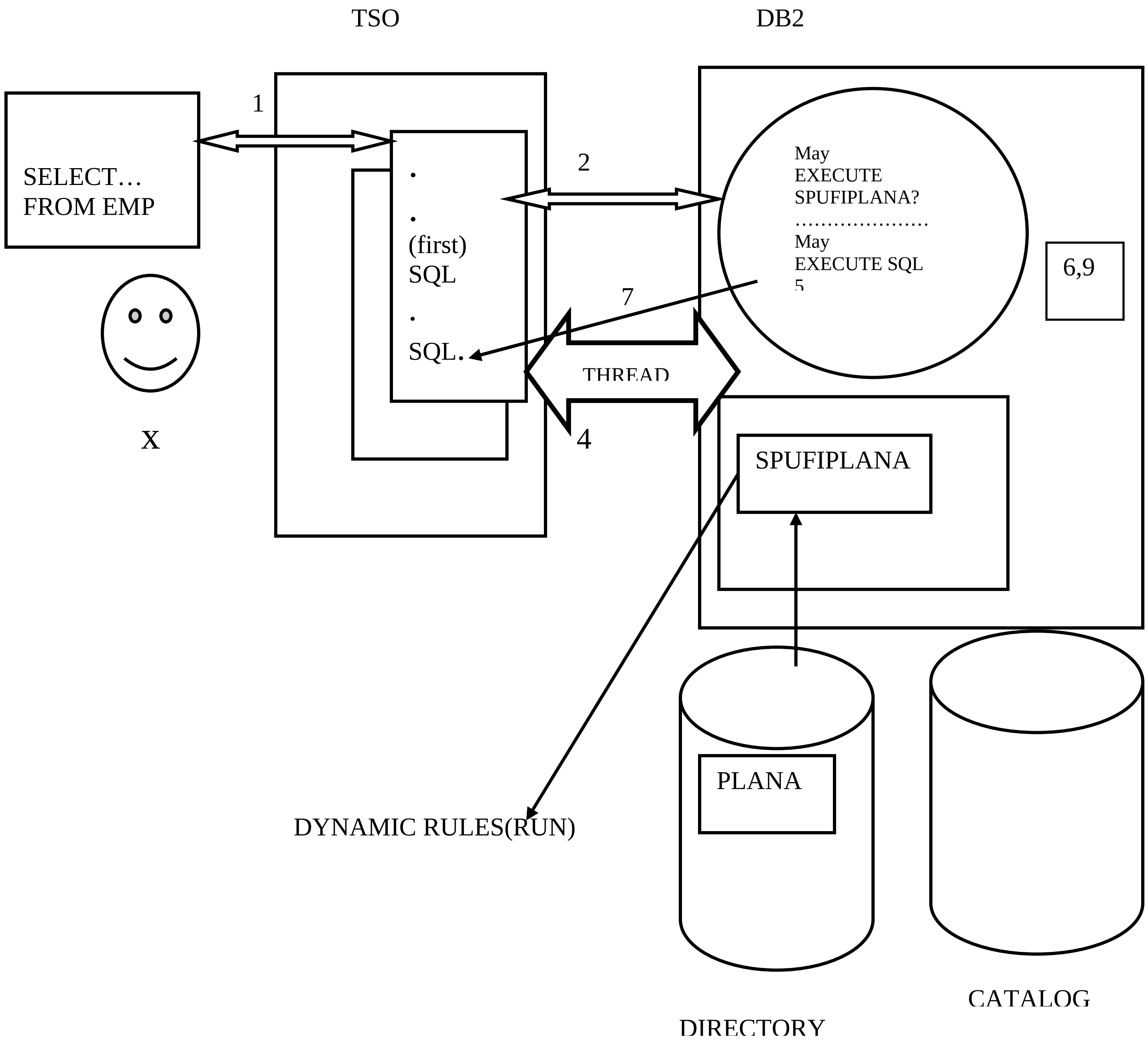


Figure 9.5 DB2 Security – DYNAMIC DQL

DB2 SECURITY - DYNAMIC SQL(Contd..)

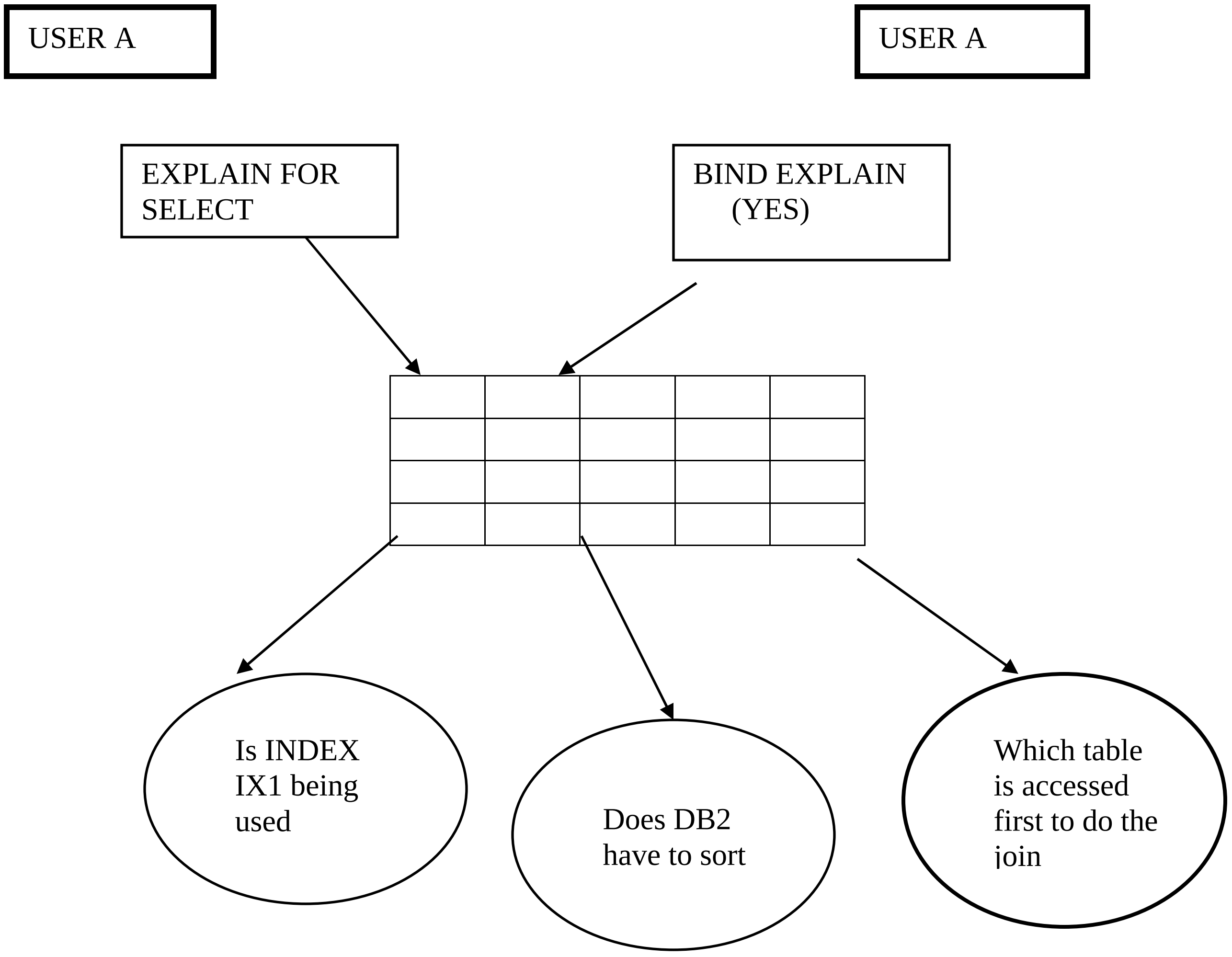
Let,s take the example of a SPUFI invocation. When a user invokes SPUFI to issue SQL statement dynamically, he will need the EXECUTE privilege on SPUFI's plan and the data access privileges on the data he wants to access :

1. User "x" logs on and invokes SPUFI. The SPUFI program is brought into storage and executed by issuing the following TSO command

   RUN PROGRAM (spufi) PLAN (spufiplan)

2. At first SQL, DB2 checks whether "spufiplan" may be executed by "x" by looking in the DB2 catalog.
3. If so, it will load "spufiplan" into storage and execute it.
4. The "thread" is created. From now on, dynamic SQL statements will be executed over the thread. The first PREPARE statement can be executed.
5. DB2 checks the data access privileges for "x".
6. Db2 chooses the optional access path to the data.
7. At the next SQL statement, data access privileges are again checked.
8. The access path is created for that SQL statement, and so on …

EXPLAIN INVOCATION



If you want to know what access path DB2 chose during a BIND or PREPARE operation, you must use the EXLPAIN function. “You can do an EXPLAIN” :

- When you BIND a package or Plan(with DBRMs) by specifying EXPLAIN(YES)
- Interactively ,by using the EXPLAIN SQL statement followed by the actual SQL statement of which you want the access path.

The EXPLAIN output consists of a number of rows in a table called the PLAN_TABLE. Each user who does EXPLAIN needs his own PLAN_TABLE.

EXPLAIN INVOCATION AT BIND TIME

```
BIND
        ………PACKAGE/PLAN
        MEMBER(DBRM1)
        EXPLAIN (YES)
```

- Insert Access path information for application SQL into Userid.PLAN_TABLE
- DB2 catalog ‘remembers’ and will redo EXPLAIN at each new BIND / REBIND until BOUND WITH EXPLAIN(NO)

The owner of the PLAN_TABLE will be the BINDer of the program.

READING THE PLAN_TABLE

For PLANs with DBRMs

```
SELECT  *  FROM  PLAN_TABLE
          WHERE APPLNA = “PLAN1”
          AND PROGNAME =”DBRM1”
          ORDER BY QUERYNO, QBLOCKNO,PLANNO, MIXOPSEQ
```

For PACKAGEs

```
SELECT  *  FROM  PLAN_TABLE
          WHERE PROGNAME = “PKG1”
          AND COLLID =”COLL1”
          ORDER BY QUERYNO, QBLOCKNO,PLANNO, MIXOPSEQ
```

Those SQL statements will come in handy if you want access path information for all the statements of a package or plan. The “ORDER BY” clauses are essential to understand the access path.

SQL EXPLAIN INVOCATION

```
EXPLAIN ALL
          SET QUERYNO = qr
          FOR SELECT A1,A2
          FROM S
          WHERE     A5=345
```

- QUERYNO is set by the user or assigned by DB2
- The EXPLAIN statement can be :
  - executed in SPUFI or QMF
  - Imbedded in an application
- The SQL statement cannot contain host variables
- Not recommended for Imbedded SQL because
  Using constants instead of host variables may change the access
  Path

The second way to “do an EXPLAIN” is interactively, as shown in the example. DB2 will, in this case, simulate a “PREPARE” to determine the access path.
As you can understand from what we have seen for dynamic SQL, host variables will have to be replaced by “?”. If you isolated an SQL statement from a program to EXPLAIN its access path interactively, you shouldn’t replace a host variable by a constant because this would mean providing more information to DB2 than it would have at BIND time. A different access path could very well be in the result, and you might draw the wrong conclusions.

COMMIT AND ROLLBACK

COMMIT and ROLL BACK are not really database operations at all, in the sense that SELECT, UPDATE are database operations.
The COMMIT and ROLLBACK statements are not instructions to the database management system. Instead, they are instructions to the transaction manager and the transaction manager is certainly not a part of the DBMS
On the contrary, the DBMS is subordinate to the transaction manager, in the sense that the DBMS is just one of possibly several "resource managers" that provide services to the actions running under that transaction manager.

- **A transaction running under CICS can also use the services of three resources managers- IMS/DB, CICS and DB2. Here CICS acts as the transaction manager.**

- **In the TSO and 'pure batch' environments, where DB2 itself serves as the transaction manager, they are requested via the explicit SQL operators COMMIT and ROLLBACK**

Before getting details of COMMIT and ROLLBACK statements as such, we first need to know "SYNCHRONIZATION POINT' abbreviated as SYNCPOINT.

A SYNCPOINT represents a boundary point between two consecutive transactions, loosely speaking, it corresponds to the end of a logical unit of work and thus to a point at which the database is in a state of consistency. Program initiation, COMMIT and ROLLBACK each establish a synchpoint and no other operation does.

Figure: 4.20 Commit and Rollback

COMMIT AND ROLLBACK (Cont...)

COMMIT :

The SQL COMMIT statement takes the form

COMMIT (WORK)

A successful end-of-transaction is signaled and a synchpoint is established. All updates made by the program since the previous synchpoint are committed. All open cursors are closed, except for cursors whose declaration includes the optional specification WITH HOLD , all rowlocks are released, except for locks needed to hold position for cursors not closed.

ROLLBACK

The SQL ROLLBACK statement takes the form

ROLLBACK [WORK]

An unsuccessful end-of-transaction is signed and a synchpoint is established. All updates made by the program since the previous synchpoint are undone.
All open cursors are closed.

Figure: 4.21 Commit and Rollback (Cont…)