Table of Contents

Overview of DB2

Reality of Digital Data

It is estimated that the amount of digital data increases at a rate of 60% each year. With 80% of the world's data processed by mainframes, organizations need to ensure that their data is stored securely and that it can be accessed reliably and quickly.
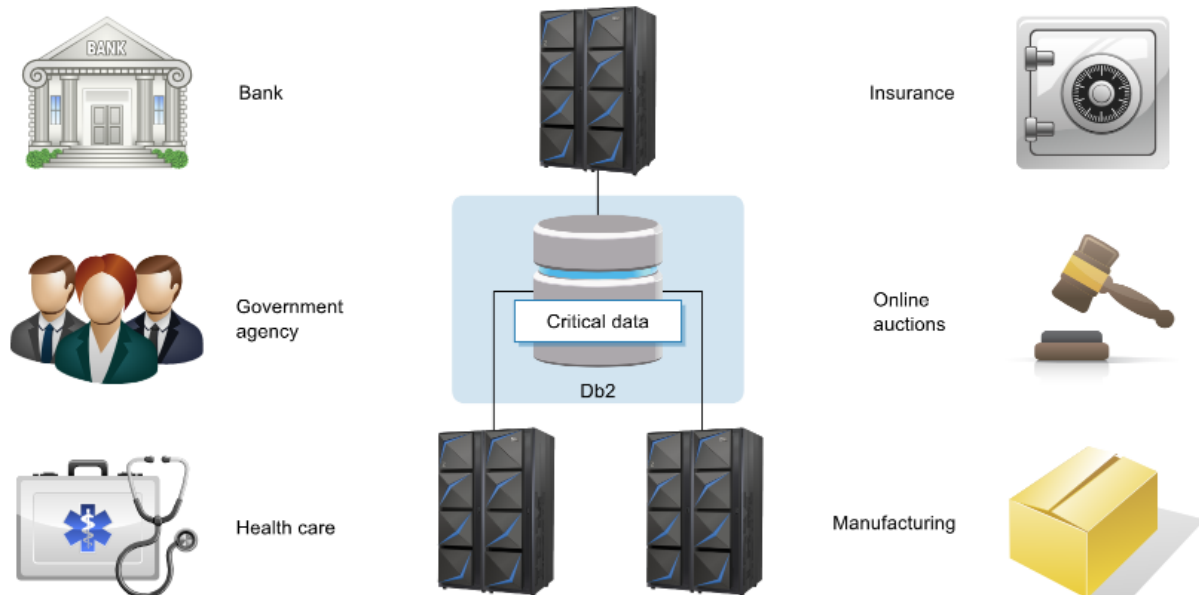
Organizations such as banks, health care, insurance and government departments need to be able to share their data between sites and applications and have it available 24 hours a day.

Introducing DB2

Many of the organisations mentioned above have turned to DB2 to manage their data. Today's DB2 can be run on a variety of platforms, but the majority of information in these notes will focus on DB2 for z/OS, which combines a robust relational database (DB2) with the dependability of the mainframe (z/OS).

Within this structure, data can be made extremely secure, while providing the organisation with continuous availability of data and a scalable solution for future data growth.

Some examples of data issues faced by organisations:
- **Banks** - May be dealing with new financial reforms requiring it to store additional records and produce new reports.
- **Health Care** - May have recently acquired a previous competitor, whose data systems are in a different format and are more geographically spread.
- **Government Agency** - May be looking to reduce energy costs and improve operational efficiency, the availability of data and access times to it.
- **Online Auctions** - May be consolidating the IT systems that are currently running different operating systems on various hardware platforms.

Advantages of DB2 for z/OS

DB2 by itself provides you with a data structure that:

- Is easy to understand. Data from DB2 databases is presented to users in table-like format. This is one of the more familiar ways of displaying data, used in, for example, telephone books and transport timetables.
- Allows design flexibility. A table of data is relatively easy to design. All you need to know are the key components of your database, such as Name, Address and Telephone number.
- Provides ease of access. The user is shielded from the internal workings of Db2. Therefore, the user does not need to know the physical relationships of the database they are accessing.

DB2 - A Relational Database

Db2 itself is a relational database, which is basically a collection of related data that is stored in multiple tables that can be reassembled to form meaningful user data.

The example displayed here shows a simple relational database structure for a healthcare organization. Three tables are displayed that contain doctor, patient, and blood test results. In reality there would be anywhere between ten and a thousand tables for this type of organization. The columns of each table form field names, while a row of data values forms a specific record.
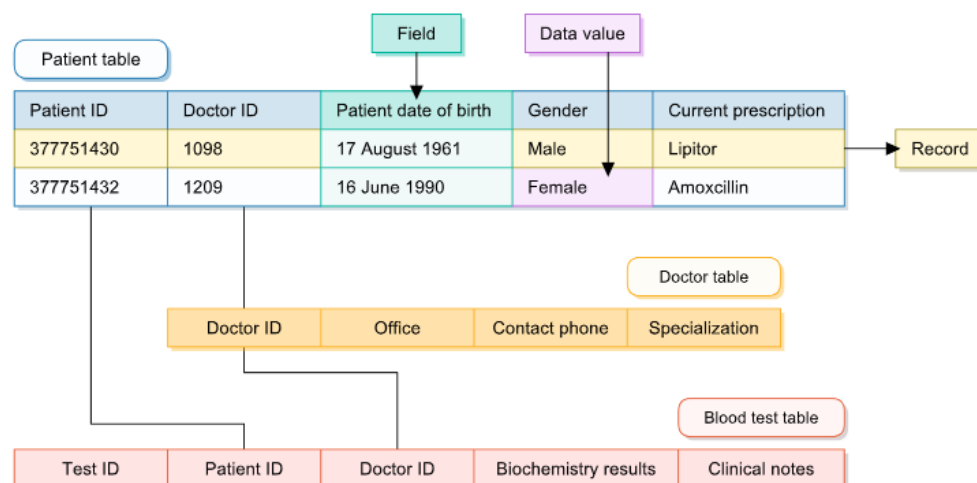
Accessing DB2 Data

The data stored in Db2 is accessed using SQL, which is a computer language specifically designed to communicate with relational databases. SQL is both simple and powerful, since a single SQL statement can generate an entire report, which would take numerous lines of conventional programming code.

As an example, if you wanted to list all doctors and sort them by the office they work in, you would need to access the Doctor table using the following SQL statement:

SELECT * FROM DOCTOR
ORDER BY OFFICE;

SQL statements can be invoked from a number of sources if the appropriate interface software to Db2 has been installed. Common z/OS environment interfaces include:
- **TSO** - Allows you to interact with DB2 using the TSO attachment facility, the call attachment facility (CAF). and the Resource Recovery Services (RRS) attachment facility.
- **Batch** - Batch jobs accessing DB2 data use the same attachment facilities as those mentioned for TSO.
- **CICS** - The CICS Transaction Server provides access to DB2 through the CICS attachment facility.
- **IMS** - This is a hierarchical database and information management system that can access DB2 data using the IMS attachment facility.
- **WebSphere** - A suite of WebSphere products have been designed to be able to access DB2 data using the Resource Recovery Services (RRS) attachment facility.

You can also start Db2 sessions from other environments on clients such as Microsoft Windows or UNIX by using interfaces like ODBC, JDBC, and SQLJ.

DB2 - Users

If a problem occurs with Db2, it may be your responsibility to identify the person it is escalated to, so it is important to understand the functions performed by IT personnel who interact with DB2.
- **Application Developer** - This person is responsible for all phases of development and maintenance associated with the development of DB2 related applications.
- **Database Administrator** - This person is responsible for the logical and physical design of databases, and implementation of those databases to support the business applications. They may also be involved in monitoring DB2 space usage and database performance.
- **DB2 Systems Programmer** - This person is usually responsible for the installation, upgrade and migration of the DB2 subsystem and related software. They may also be responsible for performance monitoring and tuning as well as reporting and resolving software issues.
- **Computer Operator** - This person monitors the overall computing system, including DB2. They may be responsible for starting and stopping the database or subsystem and responding to system messages produced by this product.

DB2 - User Groups

If you are working with DB2, then there are a considerable range of options available to keep up to date with DB2 developments and interact with people.

User groups, and their websites and resources, can benefit you by expanding your resource pool and provide you with exposure to technical personnel with common areas of interest. **The International DB2 Users Group** is one such organization, whose website contains information on conferences, blogs, and webcasts.

Many individual states have formed their own DB2 user groups, making it easier to attend meetings and meet people face to face.

DB2 - Publications

There are a number of sources that provide DB2 users with publications such as technical user guides, useful tips, case studies, and general articles on various DB2 products and utilities.

IBM's online Knowledge Center is always a great place to start when looking for in-depth information, while IBM's DB2 product page contains links to articles and blogs on DB2 products.

The DB2 Environment

DB2 - Components

As mentioned previously, DB2, or the DB2 data server as it is more commonly known, and its associated products can be implemented and run on a number of different operating systems including: z/OS, IBM i, Linux, UNIX, and Microsoft Windows. This is useful if an organization consists of a mixture of these systems.

While each version shown here is tweaked to support the individual capabilities of that system, SQL is virtually the same in each environment, allowing data sharing to occur between data servers and application code with minimal modifications.

DB2 - Capabilities

Db2 data servers support a large number of different clients and languages, and with additional support can integrate data from other sources.

Data sources	Languages	Clients
IMS	APL2	AIX
Informix	Assembler	Eclipse
Oracle	C	HP-UX
Microsoft® SQL Server, Excel	C++	Linux
Sybase	C#	Solaris
JDBC	COBOL	Windows
Databases that support the JDBC API	Fortran	Web browsers
OLE DB	Java	
Teradata	.NET	
EMC Documentum	Perl	
	PHP	
	PL/I	
	Python	
	REXX	
	Ruby on Rails	
	SQL procedural language	
	Visual Basic .NET	

DB2 - Tools

There is a long list of tools that may be implemented by your organization to support aspects of DB2 for z/OS performance, administration, backup and recovery, automation, and replication and application management. DB2 for z/OS includes a comprehensive toolset, but many additional tools are optionally available from IBM as well as from third-party software vendors.

The tools mentioned below are provided by IBM.
- **DB2 Utilities Solution Pack for z/OS V4.2**
  - This solution provides you with the framework to better execute and manage DB2 utilities, ensuring improved performance optimisation and resource utilisation. The DB2 Automation Tool for z/OS can be set up to work with the Data Server Manager.
- **DB2 Performance Solution Pack for z/OS V1.5**
  - This solution enables you to quickly identify, diagnose, solve and prevent performance problems in your DB2 for the z/OS environment.
- **DB2 Administration Solution Pack for z/OS V3.1.0**
  - This solution manages the complexity, growth, change and cloning of DB2 for the z/OS objects and scheme throughout the application lifecycle.

Many of the DB2 for z/OS tools display data using a graphical user interface or ISPF and allow you to perform a number of DB2 tasks interactively.

Middleware

Today's on-demand environment requires powerful and efficient middleware products and APIs to define and manage application development and facilitate communication between clients and the Db2 content.

Doctor ID	Office	Contact phone	Specialization
1198	Anaheim	(562) 933-2000	Dermatologist
1202	San Diego	(619) 287-3270	Cardiologist
1203	Berkeley	(510) 204-4444	Anaesthetist
1313	Berkeley	(510) 204-2579	Oncologist
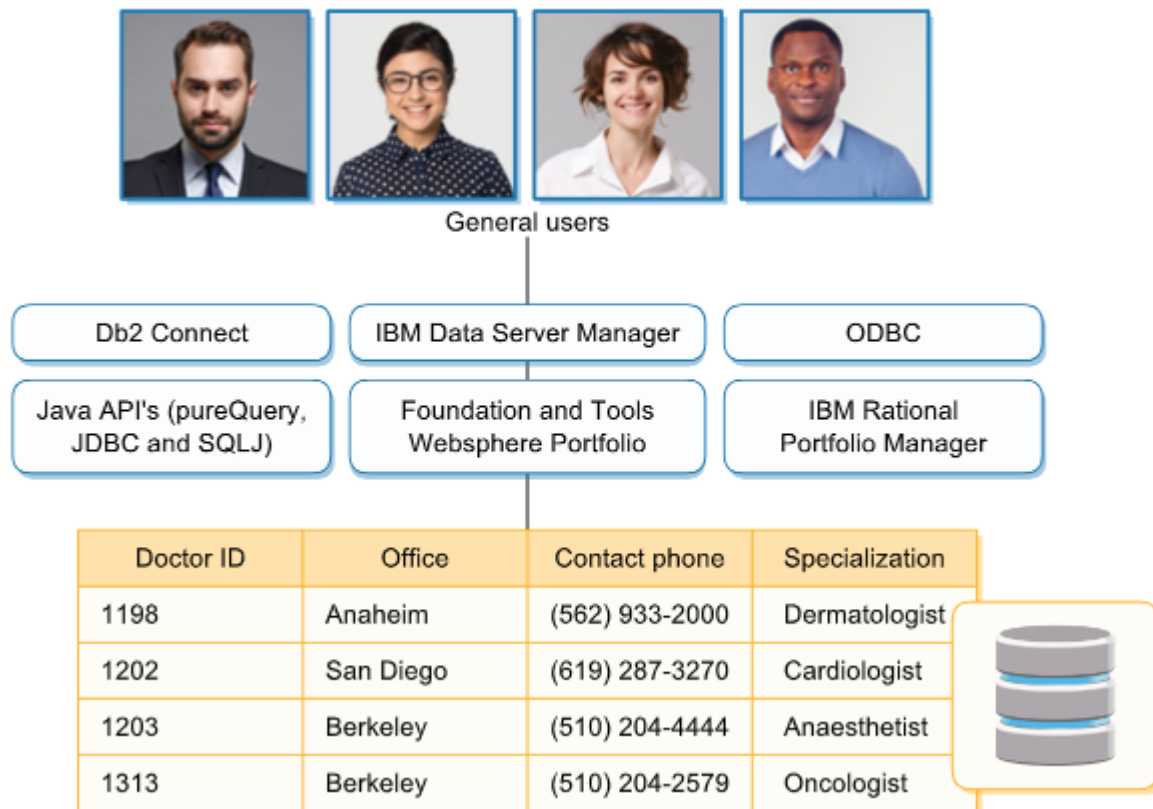
Accessing DB2 Data

Structured Query Language (SQL)

In the scenario shown here, an application programmer has coded some structured query statements to access specific DB2 data. SQL consists of over 100 different statements that can be used to insert, query, update, delete, and authorize access to DB2 data.
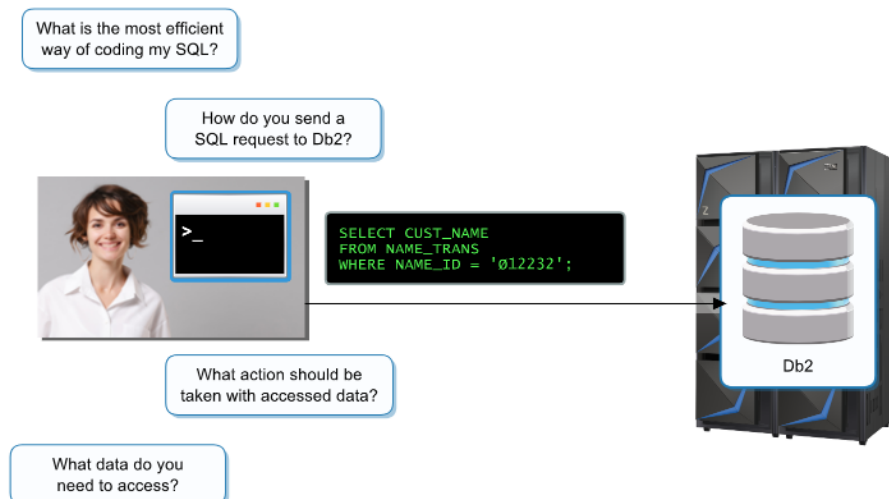
Embedding SQL in an Application Program

SQL statements can be inserted into source application programs. This type of invocation is referred to as an embedded statement.

In the COBOL example shown here, the keywords EXEC SQL are required to invoke SQL statements and END-EXEC indicates the end of the SQL stream.
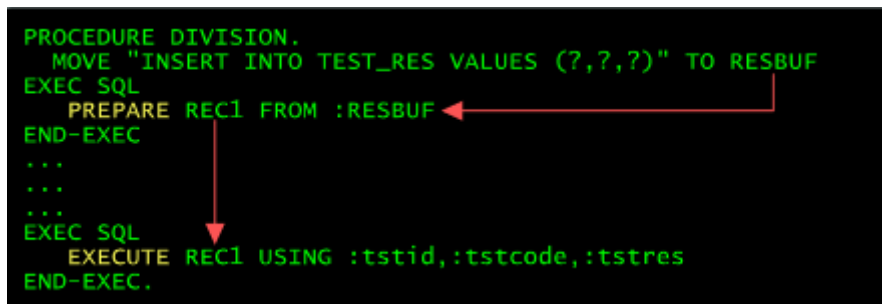- For C and C++ programs, each SQL statement must begin with EXEC SQL and end with a semicolon (;).
- For REXX, each SQL statement must begin with EXECSQL (in either upper-, lower-, or mixed-case) and be followed by either an SQL statement (enclosed in single or double quotation marks) or a REXX variable that contains an SQL statement.
- For Java, embedded SQL is executed within SQLJ executable clauses which begin with #sql and end with a semicolon.

```
    EXEC SQL DECLARE LRNR.DEPARTMENT TABLE
    ( DEPTNO                        CHAR(3) NOT NULL,
      DEPTNAME                      VARCHAR(29) NOT NULL,
      MGRNO                         CHAR(6),
      ADMRDEPT                      CHAR(3) NOT NULL,
      LOCATION                      CHAR(16)
    ) END-EXEC.
*****************************************************************
* COBOL DECLARATION FOR TABLE LRNR.DEPARTMENT                   *
*****************************************************************
 01  DCLDEPARTMENT.
     10 DEPTNO              PIC X(3).
     10 DEPTNAME.
        49 DEPTNAME-LEN     PIC S9(4) USAGE COMP.
        49 DEPTNAME-TEXT    PIC X(29).
     10 MGRNO               PIC X(6).
     10 ADMRDEPT            PIC X(3).
     10 LOCATION            PIC X(16).
```

Dynamic Preparation of SQL

Rather than using static SQL statements as in the previous example, you can dynamically build SQL statements based on input provided to the application.

```
PROCEDURE DIVISION.
   MOVE "INSERT INTO TEST_RES VALUES (?,?,?)" TO RESBUF
EXEC SQL
   PREPARE REC1 FROM :RESBUF  ◄──────
END-EXEC
...
...
...
EXEC SQL
   EXECUTE REC1 USING :tstid,:tstcode,:tstres
END-EXEC.
```

The example here shows that PREPARE and EXECUTE statements have been used to create an executable SQL statement from a string and then execute it.
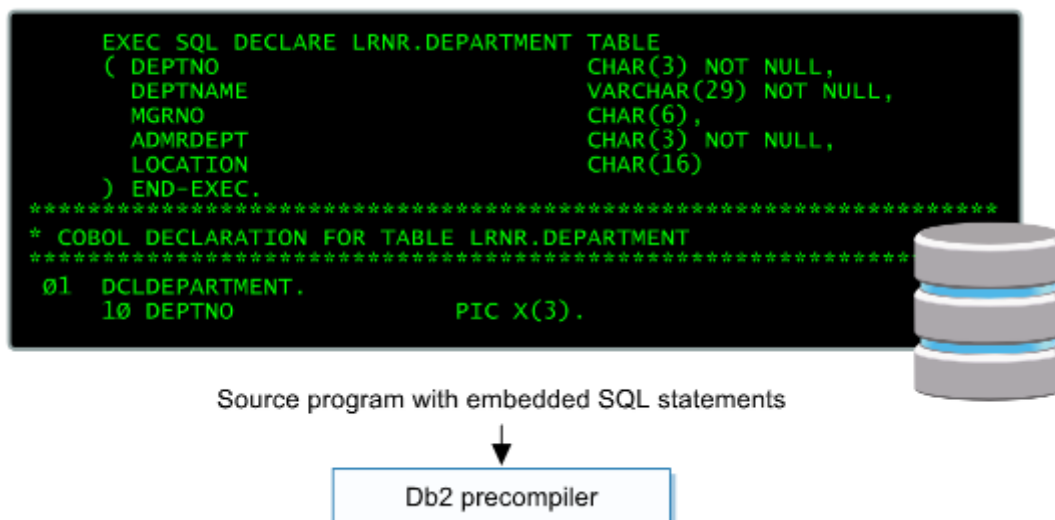- All non-Java languages use this code or the EXECUTE IMMEDIATE statement.
- While Java uses Statement, PreparedStatement, and CallableStatement classes to perform the same function.

Preparing the Source Program

In traditional programming languages, whether there are static or dynamic SQL statements in your source program, they both need to be processed before the program is compiled.

With **Static SQL** statements, the DB2 precompiler or coprocessor checks the syntax of the SQL statements and if acceptable will convert them into host language statements that are used to access DB2 data.

Dynamic SQL statements are handled differently in that they are not precompiled but are instead passed to DB2 as a character string by the program. These statements are then processed at run time.

```
        EXEC SQL DECLARE LRNR.DEPARTMENT TABLE
        ( DEPTNO                         CHAR(3) NOT NULL,
          DEPTNAME                       VARCHAR(29) NOT NULL,
          MGRNO                          CHAR(6),
          ADMRDEPT                       CHAR(3) NOT NULL,
          LOCATION                       CHAR(16)
        ) END-EXEC.
**************************************************************
* COBOL DECLARATION FOR TABLE LRNR.DEPARTMENT
**************************************************************
  01  DCLDEPARTMENT.
      10 DEPTNO              PIC X(3).
```

Source program with embedded SQL statements

Db2 precompiler

The DB2 precompiler stage produces a Database Request Module (DBRM) that contains the SQL statements and host variable information from the source program. This information is used when the BIND command is invoked, and will create a plan or package that contains control structures and processing options.
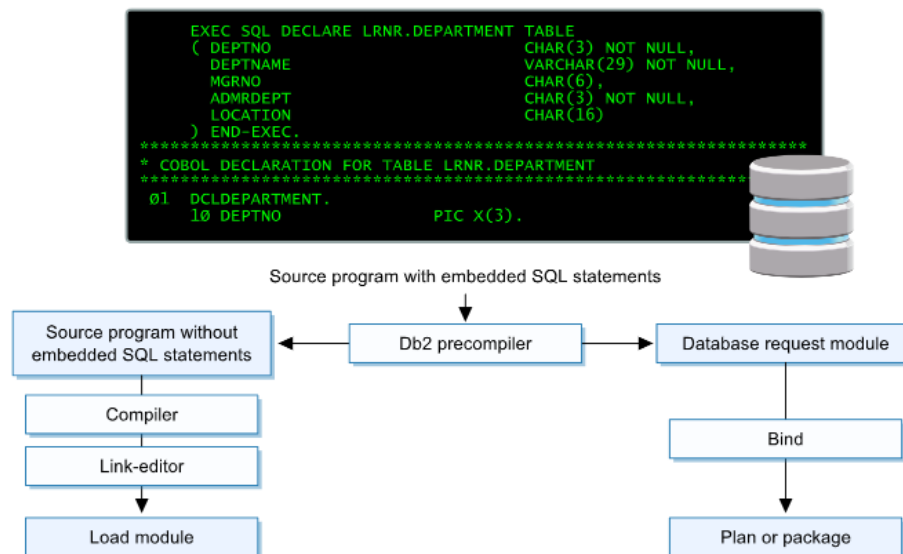

Finally an executable load module needs to be created from your source program using a link-edit procedure. The DB2 application is now ready to run.

Stored SQL Procedures

Where an application program containing SQL statements performs a common function, it can be placed as a stored procedure on the database server. The stored procedure is then invoked from an application program using the CALL statement. The benefits of having a stored procedure are the following:

- There is less network traffic as the procedure is stored in close proximity to the data.
- Less duplication of code as it is only written once, which means that any changes will also only need to be made to one set of code.
- Security can be enhanced by assigning database privileges to the stored procedure rather than to users.
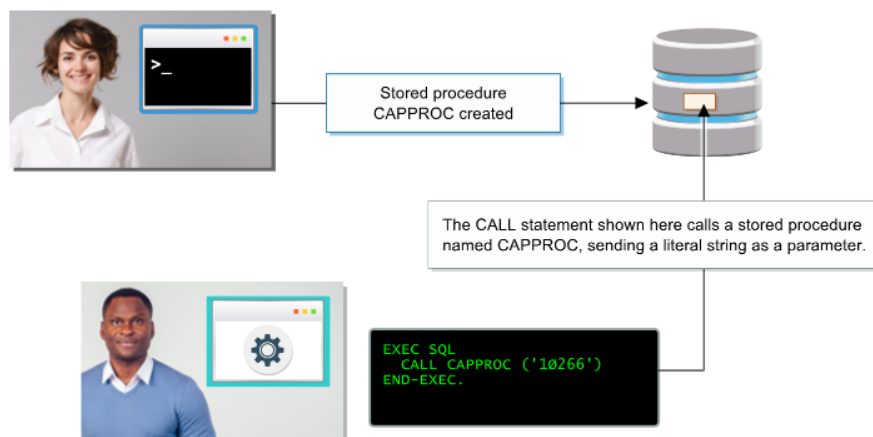
Interactively Invoking SQL

SQL statements can be sent to DB2 directly from a workstation. In the example shown here, a TSO facility called **SPUFI (SQL Processor Using File Input)** allows you to execute SQL statements from an input file, without needing to embed them in an application program.

XML Data

XML data can also be stored in a DB2 table and is accessed, with a few exceptions, using the same SQL statements. This XML data can be used by organizations for document processing and providing information through its Web services.
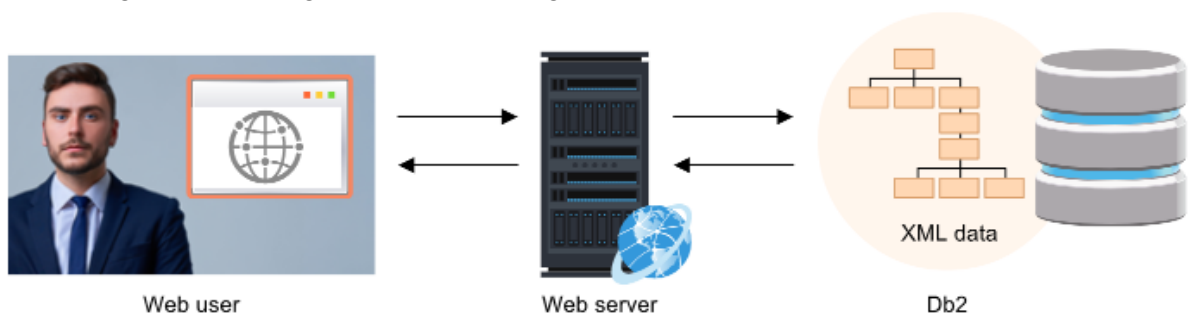
DB2 uses a feature called **pureXML** to manage XML data stored in DB2 tables.

JSON

Today's rapidly changing application environments have seen the emergence of JavaScript Object Notation (JSON) as a key technology to achieving an organization's mobile, social, big data analytics, and cloud data needs.

New web applications often use JSON for storing and exchanging information. With DB2 V10 for z/OS, document storage support for JSON was added, relying on DB2 NoSQL JSON APIs.

DB2 V11 for z/OS added two additional SQL interfaces, User Defined Functions:
- One to convert JSON to BSON
- The other is to convert BSON to JSON.

These allow SQL to be used to perform basic manipulation of JSON data. Db2 V12 for z/OS improves support for web and mobile applications with two ways of working with JSON: the Java driver for the JSON API or via SQL extensions.

The example shown here identifies various methods in which JSON documents can be queried and managed.
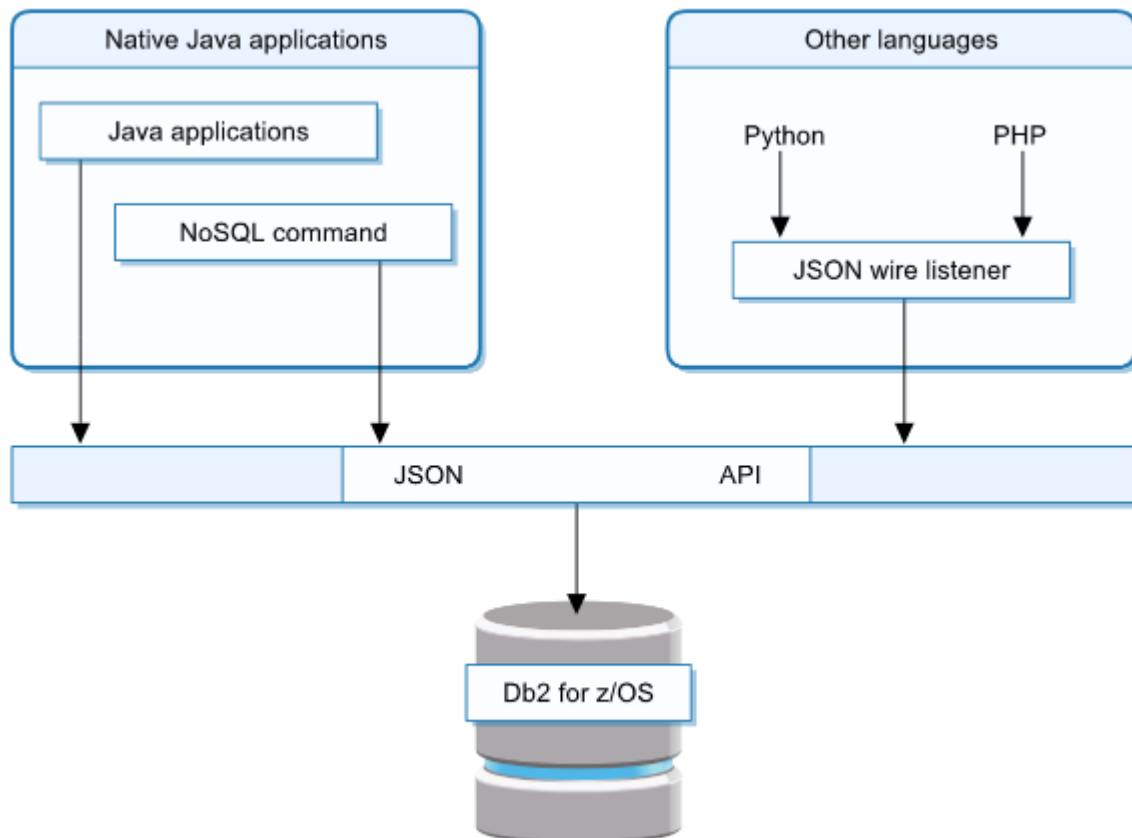
DB2 Data Components

Tables

One of the most basic structures within DB2 is the table, which contains data your organization needs to store. A primary key is used when you need to create a referential integrity relationship with a foreign key in another table. If creating a primary key, it must have a corresponding index created, the process of which you will see shortly.

There are several types of tables that can be created from the commonly used base table, result tables, sample tables, temporary tables, history tables, auxiliary tables, and others.

Table Spaces

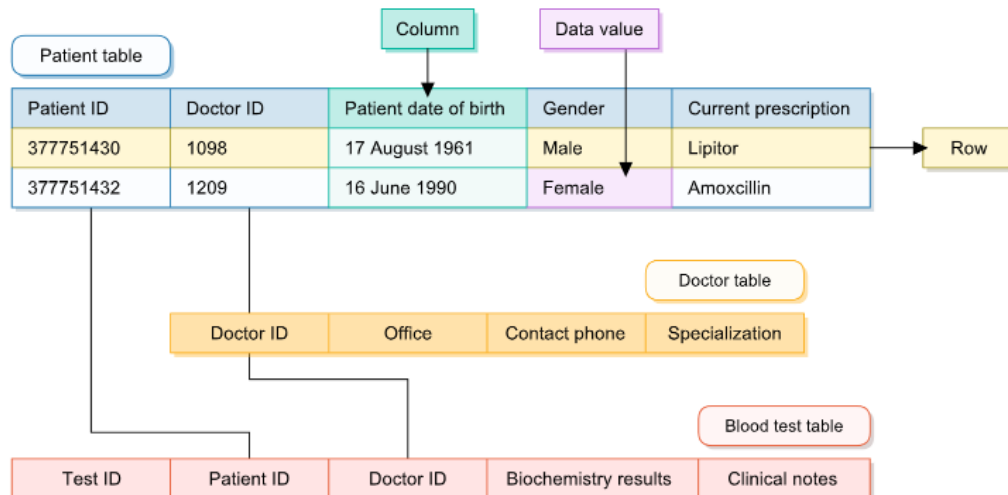All tables within DB2 are stored in table spaces, which is a storage structure that consists of **VSAM data** sets. Table spaces can be created by issuing a **CREATE TABLESPACE** command or by DB2 automatically when a table is created.
Types of table space that DB2 can work with include:
- Partition-by-growth universal table spaces (PBG UTS)
- Partition-by-range universal table spaces (PBR UTS)
- Segmented non-UTS table spaces (deprecated)
- Partitioned non-UTS table spaces (deprecated)
- Simple table spaces (deprecated)
- Large object table spaces
- XML table spaces

Note: As of DB2 12 for z/OS, non-UTS table spaces for base tables are deprecated and are likely to be unsupported in the future.

There are many types of table spaces that can be defined with the type you use being dependent on the type of data being stored and the amount of space available.

Table space			
Doctor table			
Doctor ID	Office	Contact phone	Specialization
1198	Anaheim	(562) 933-2000	Dermatologist
1202	San Diego	(619) 287-3270	Cardiologist
1203	Berkeley	(510) 204-4444	Anaesthetist

Indexes

Just like an index for a book, DB2 indexes are used to quickly locate information without having to read all the content. In the example from the previous section, an index could be created for the Doctor ID column to locate a specific doctor's record.
The main benefit derived from using an index is performance, because not all data in the table needs to be accessed during a request.

Index Spaces

Just as tables have their own table space defined, so do indexes. A separate index space is created for every index.
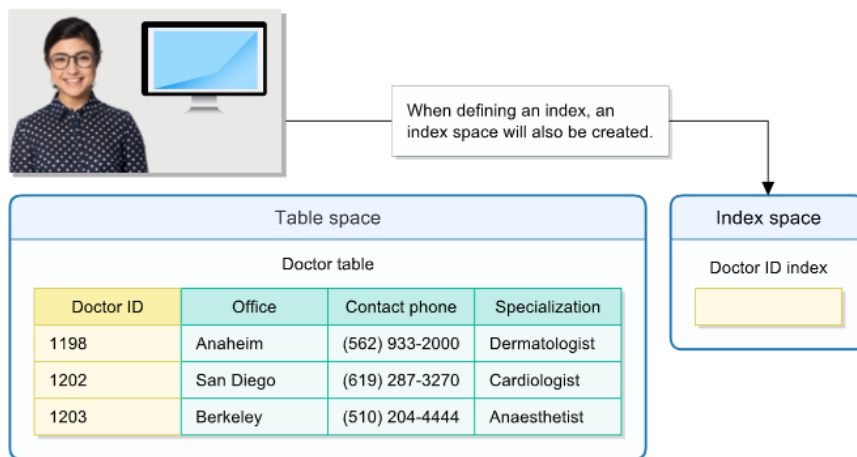
Keys

Tables can, and often do, contain columns that are defined as keys. These keys are used to ensure that each record in the table is unique, for example, that there are no duplicate doctor IDs, and that data from one table cannot be removed if it has related data in another table.

You will often see a primary key defined for a table, which is a unique key that is not allowed to contain null values. An index can be created for this type of key and is called a primary index. DB2 also supports a number of other keys including unique keys, parent keys, foreign keys, and composite keys.
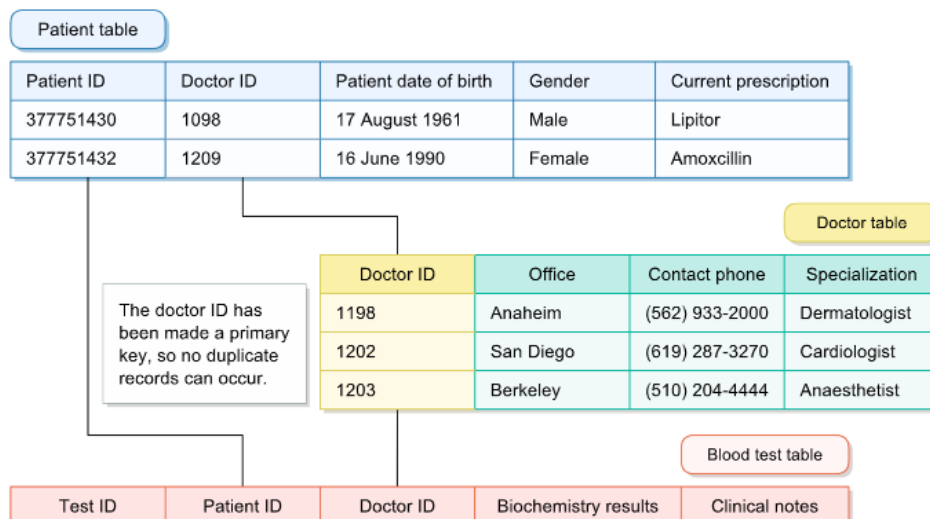
Views

There may be occasions where specific data from several tables is often requested, or users are not authorized to view certain data in a table. In these scenarios, views can be created which are subsets of existing data.

When a request for a new view is issued, only the definition supporting that view is stored. The data used to make up the view is already in existing tables, so creating separate data in views would result in much data duplication..

This user can only access a view of the table.

Storage Groups

A storage group is a defined area on disk used to store tables and indexes. When these components are created you can specify the storage group whose space is to be utilized.

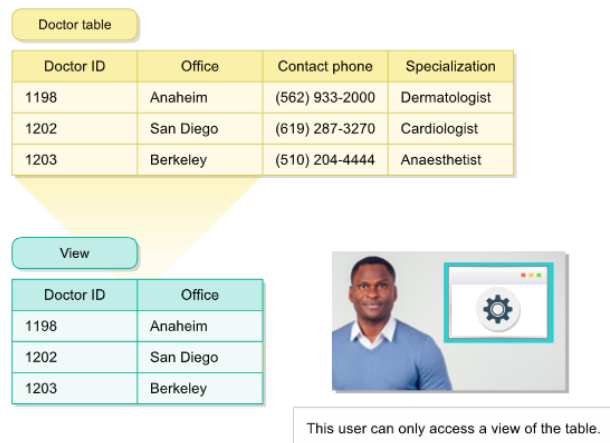**Note:** These storage groups are not the same as the ones used by DFSMS.
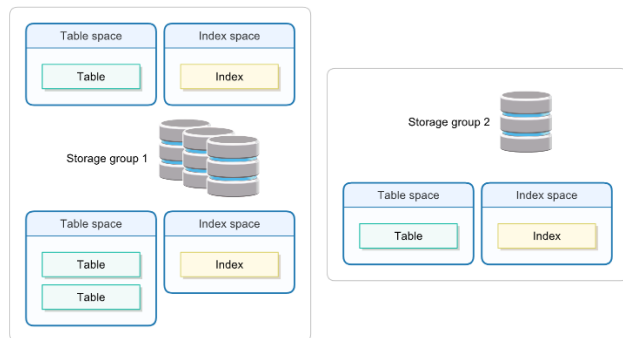
Schemas

A schema is used to group a set of objects within the Db2 database together using a high-level qualifier name. For example, if a schema called Accounts is created and there are two tables in the database, Cust and Invoice, that are linked to the schema, then the fully qualified names of the tables are Accounts.Cust and Accounts.Invoice.

Schemas can also consist of indexes, table spaces (note that the schema for a table space is actually the database it belongs to), functions, stored procedures, and triggers. Whenever any of these elements is created, it is automatically assigned a high-level qualifier schema name.
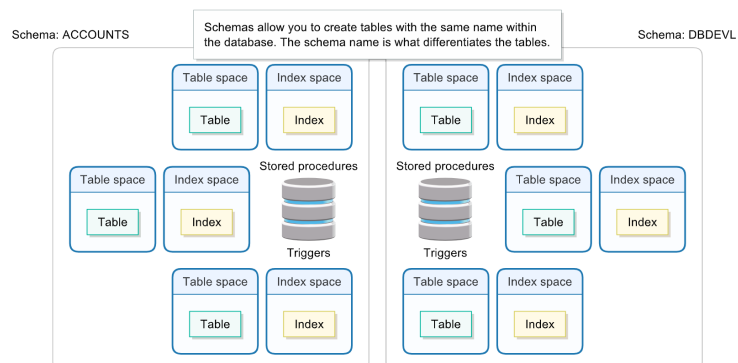
Databases

All of the data components you have looked at so far are linked together to form a database. Linking them like this enables you to control an entire application by entering commands to the database entity. For example, you may need to prevent access to all data relating to that database because there is a problem, or you may want to grant access to a user that needs to use the information in that database.

Database: STATCE1

Table space	Index space
Table	Index
Table space	Index space
Table	Index
Table space	Index space
Table	Index

-STOP DATABASE(STATCE1)

```
REVOKE CREATETAB
  ON DATABASE D99GIDE
  FROM GHAM;
```

Table space	Index space
Table	Index
Table space	Index space
Table	Index

Database: D99GIDE
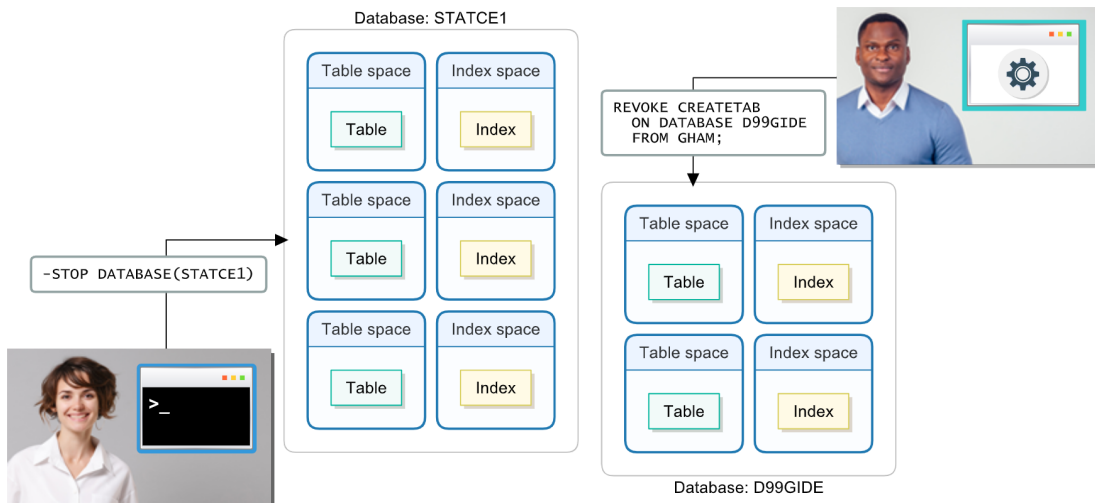
DB2 System Components

DB2 - Catalog

DB2 keeps track of objects such as tables, table spaces, indexes, index spaces, views, and storage groups within the DB2 database using a catalog. The catalog itself is a set of tables that contains details of when the objects just mentioned are created, modified, or deleted.

The catalog stores the name of the table, as well as its table space, its owner, the name of the database, and authorization information. It also stores information about the defaults relating to each column within the table.
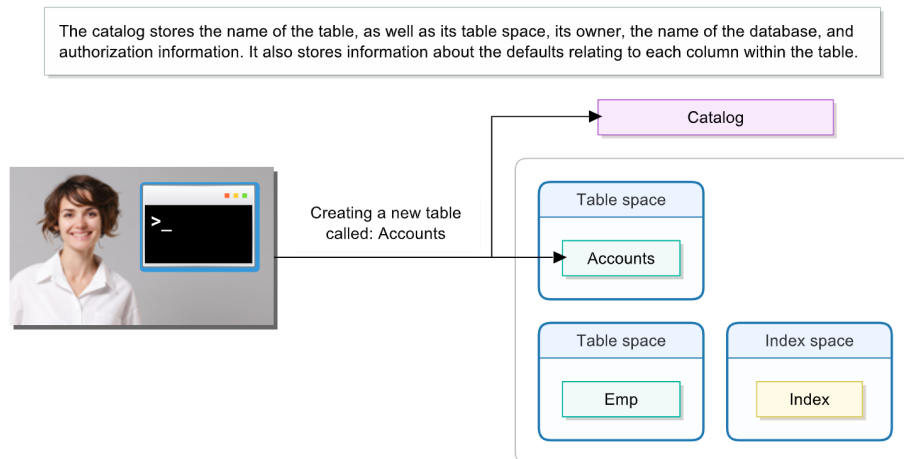
Because the catalog is a set of tables, it can be accessed using SQL.

DB2 - Directory

The DB2 directory consists of a number of directory tables stored in database DSNDB01. These tables contain system related information that is designed to be used internally by DB2, but a system administrator with appropriate authority can access data from them using SQL.

The information stored in the directory also assists with recovery and restart processes.
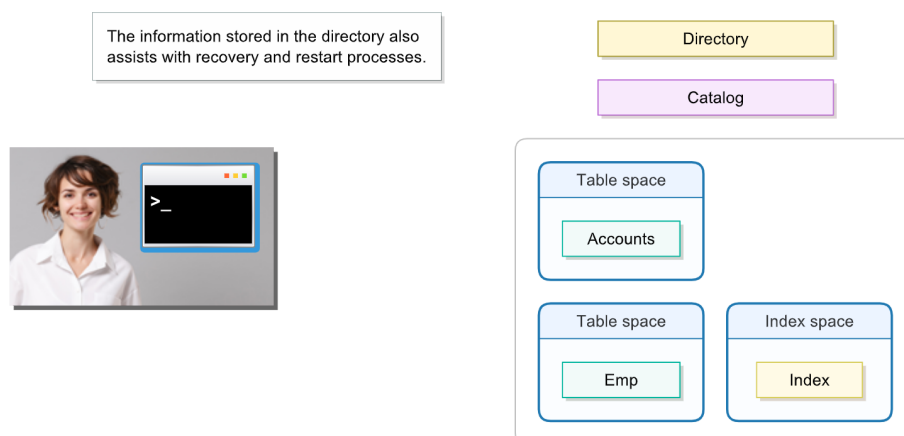
For example, details of active or stopped utility jobs can be obtained from the directory.

DB2 - Logs

Like most products, DB2 stores information relating to data changes and major events in a log. During normal operation, an active log copy will have a number of data sets available for storing this data, and when one of these active log data sets becomes full, it is automatically backed up to tape or disk and is referred to as an archived log.
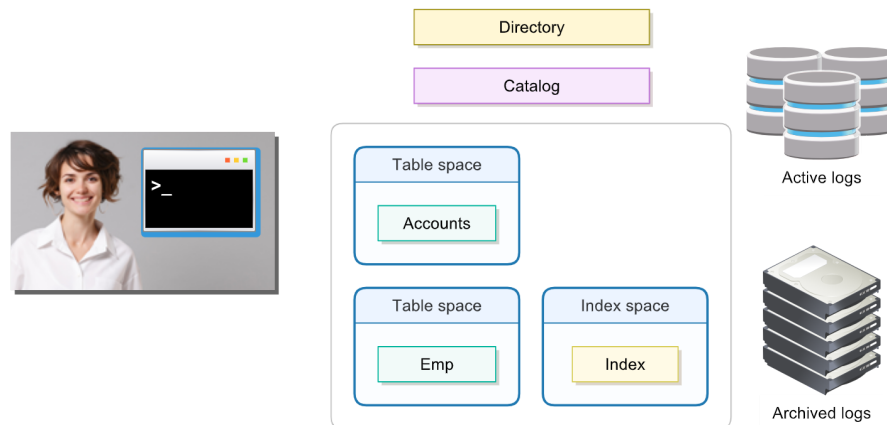
The logs can be browsed to help analyze problems and are also used in recovery scenarios to roll changes backward or forwards to a point where the database was known to be consistent.

DB2 - Buffer Pools

When an application program needs to use Db2 table or index data, that data is accessed and stored in a temporary area called a buffer pool. Access to data in this buffer pool is much quicker than if the application program had to request it again from the table or index on DASD.
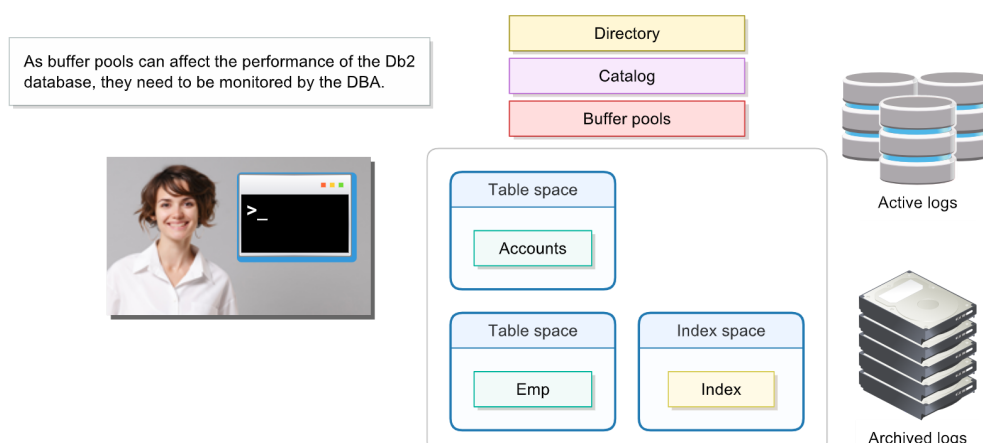
The data in the buffer pool can be read or modified and is copied back to the table or index following the completion of processing.

Databases

Several related databases can be populated and used to provide support for DB2. For example:

- Data Definition Control Support Database - This database contains information that is used to prevent certain data definition language statements from being submitted.
- Resource Limit Facility - This consists of one or more tables that define the amount of processor resources that can be used by certain types of SQL statements.
- Work File Database - If SQL statements require additional space for processing, this database can be used.
- Communications Database - This database is used to communicate with remote database management systems.

DB2 and the Parallel Sysplex

If an organization connects z/OS systems using a Parallel Sysplex then it will probably want to configure DB2 so that it can share data across the network.
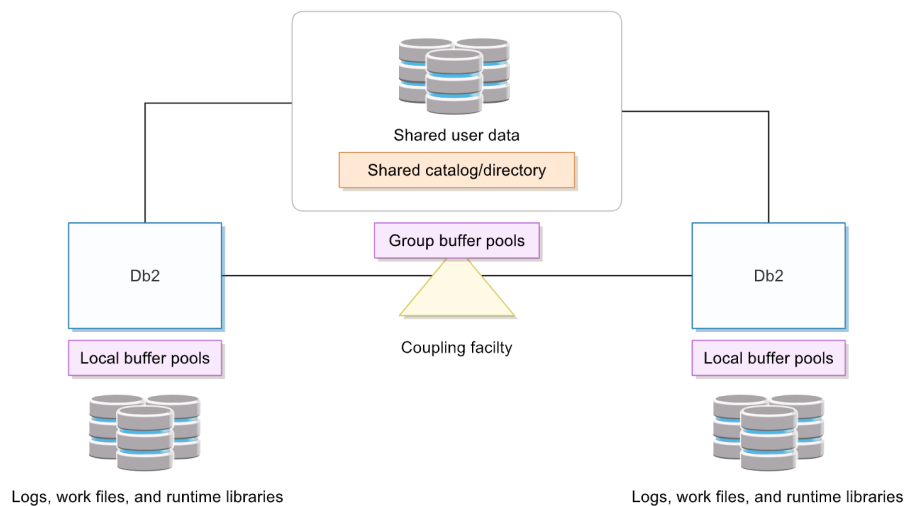
The graphic displayed here shows how such a configuration can look and the data that is shared.

Introduction to SQL

SQL is used to obtain and manipulate data that is stored in DB2 tables. SQL consists of over 100 different statements that can be used to insert, query, update, delete, and authorize access to DB2 data.

Invoking SQL

SQL statements can be inserted into source application programs and invoked when they are executed or they can be run interactively using a TSO product such as SPUFI. The examples displayed in this section are created using this method.

SQL Syntax

When coding SQL you need to adhere to the syntax rules, otherwise your code will not execute.

These are the names of SQL statements and are coded at the beginning of the statement.

Most SQL statements contain optional and required items that can be entered for that statement. In this example, the SELECT statement is used to specify the names of the table columns from which data will be obtained.

Commas are used to separate values, which in this case is column names.

```
SELECT LASTNAME, FIRSTNME
FROM EMP
WHERE DEPT = 'D11'
ORDER BY LASTNAME;

CREATE VIEW PRJ_MAXPER -- For support personnel
```

It is common to see SQL statements being terminated with a semi colon.

Generally SQL is not case sensitive, but there are several statements that do require that the correct case is used. The IBM Db2 for z/OS SQL Reference manual should be accessed for this information.

Comments can be specified by either using a double hyphen or by starting a line with the /* characters and ending the comment with */.

SELECT Statement

One of the most commonly used SQL statements is **SELECT**. This statement is used to query a table and produce results that are in a table format. The simple example here selects and displays all entries from the ADDR01.TABLE table.

```
BROWSE USER01234.RESULT  ---------------------------------Columns 001 072
COMMAND INPUT ==>                                    SCROLL ===> PAGE
---------+---------+---------+---------+---------+---------+---------+---------+
SELECT *                                                    000100000
FROM ADDR01.TABLE;                                          000200000
---------+---------+---------+---------+---------+---------+---------+---------+
LASTNAME       FIRSTNME      ADDRESS                              PHONE   GENDER
ADDISON        PAUL          123 RUSHDALE STREET, COOTAMUNDRA     4819    M
BAKER          GAIL          456 SMITH STREET, EASTWOOD           5767    F
DAVIS          MICHELLE      12 TINTERN AVENUE, ASHFIELD                  F
D'SILVA        MARK          25 HIGH STREET, SEAFORD                      M
FISHER         PHILLIP       2/144 POWER STREET, HAWTHORN                 M
MACKINTOSH     HARRY         154 SYDNEY DRIVE, SWAN VIEW          2236    M
PETERSON       BRETT         16 CHARLIES PLACE, TOOWOOMBA         7673    M
RICHARDSON     PETER         97 STURT HIGHWAY, GLENELG            4231    M
SMITH          EYVONNE       17/125 GRANGE ROAD, SOUTH YARRA      6582    F
SMITH          GREG          49 EVANS ROAD, PENRITH               7164    M
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE610I NUMBER OF ROWS DISPLAYED IS 10.
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0.
---------+---------+---------+---------+---------+---------+---------+---------+
```

SQL statement that was entered

Table data

General invocation messages

Using the table from the previous example, if certain columns of data only needed to be retrieved, then just those column names can be specified as shown in this example.

Clause Statements

When the data you require needs to be further qualified, several types of clauses can be used with the SELECT statement.

- The **WHERE** clause allows you to select data from rows that meet certain conditions. In this example, all columns will be displayed for those rows which contain the value F in the column named GENDER.
- The **ORDER BY** clause allows you to sort the rows you have selected in a particular order. For example, if you wanted a listing of all rows in your EMP.TABLE table, sorted by the department they work in, then the SQL statement shown below can be used.
- Once specific data has been requested from a table you may want to group the records together. In the example below, the OFFICE and the average of all the SALARY values is selected and then those rows will be **GROUPED BY** the office.
- The **HAVING** clause is used in conjunction with the GROUP BY clause to further refine the data to be displayed. In this example, the MAX function is used to return the name of each TESTID and maximum cost of that test. The HAVING clause will only display those TESTIDS that have a maximum cost greater than 500.
- The amount of data displayed by your SELECT request can be limited by using the **FETCH FIRST** clause. This is useful if you are working with large tables and do not require all the data or are performing testing and need only a subset of data.In this example, only the first 20 rows of the table are selected.

```
SELECT * FROM ADDRØ1.TABLE
    WHERE GENDER = 'F';
```

```
SELECT * FROM EMP.TABLE
    ORDER BY DEPT;
```

```
SELECT TESTID, MAX(COST) FROM TEST.TABLE
            GROUP BY TESTID
            HAVING MAX(COST) > 5ØØ;
```

```
SELECT OFFICE, AVG(SALARY) FROM DOC.TABLE
            GROUP BY OFFICE;
```

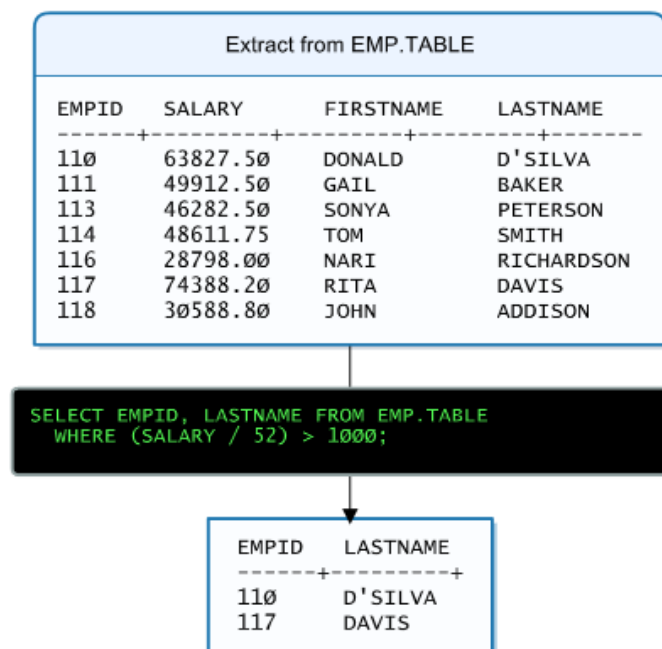```
SELECT * FROM EMP.TABLE
FETCH FIRST 2Ø ROWS ONLY;
```

## Arithmetic Operators

There will be times when you need to select rows from a table based on a mathematical calculation.

For example, you may need to select all the employees in the organization that get paid more than $1000 a week.

Arithmetic operators such as plus (+), minus (-), multiply (*), and divide (/) can be used with any column name, whether or not they are specified to be displayed in the SELECT statement, provided that the column's data type is numeric.

```
Extract from EMP.TABLE

EMPID     SALARY        FIRSTNAME     LASTNAME
------+---------+---------+---------+-------
110       63827.50      DONALD        D'SILVA
111       49912.50      GAIL          BAKER
113       46282.50      SONYA         PETERSON
114       48611.75      TOM           SMITH
116       28798.00      NARI          RICHARDSON
117       74388.20      RITA          DAVIS
118       30588.80      JOHN          ADDISON
```

```
SELECT EMPID, LASTNAME FROM EMP.TABLE
  WHERE (SALARY / 52) > 1000;
```

```
EMPID     LASTNAME
------+---------+
110       D'SILVA
117       DAVIS
```

Comparison Operators

There have been several examples already shown in this section where values are compared to produce a subset of the original table data. SQL supports a number of different types of comparison operators that can be used.

```
SELECT * FROM BUSNESS.TABLE
  WHERE CUSTNAME = 'GH LIMITED';
```

```
SELECT ORDER, COST, MARKUP FROM SALES.TABLE
  WHERE MARKUP > 100;
```

```
SELECT INGRED, SHOP, COST FROM RECIPE.TABLE
  WHERE COST <= 20;
```

The following comparison operators can be used:	
=	Equal to
<>	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

23

Predicates

Another item that can be added to a SQL statement is a predicate. A predicate is used against a row or group and returns a value of true, false, or unknown. Where the result is true, that data is selected for the result table.

- **Scenario 1** - You need to select employees from a table who have a salary between 30,000 and 40,000.

  SELECT EMPNAME, WAGE, DEPT FROM EMP.TABLE
     WHERE SALARY BETWEEN 30000 AND 40000;

- **Scenario 2** - You only require names to be selected from a lastname column in an address table.

  SELECT DISTINCT LASTNAME FROM ADDR01.TABLE;

- **Scenario 3** - You need to display all patients that live in a specific area where at least one of those patients has returned a specific test result.

  SELECT PATID, AREA, TESTRES FROM CUST.TABLE A
     WHERE EXISTS (SELECT * FROM CUST TABLE
     WHERE A.AREA=AREA AND TESTREST='K108');

- **Scenario 4** - You need to compare results from a SELECT statement with one or more values.

  SELECT CUSTNAME, CUSTID, FROM ORDER.TABLE
     WHERE CUSTID IN ('1079', '1220', '1378')
     ORDER BY CUSTNAME;

- **Scenario 5** - You need to display all clients that live in a suburb with SOUTH in its name.

  SELECT * FROM CLIENT.TABLE
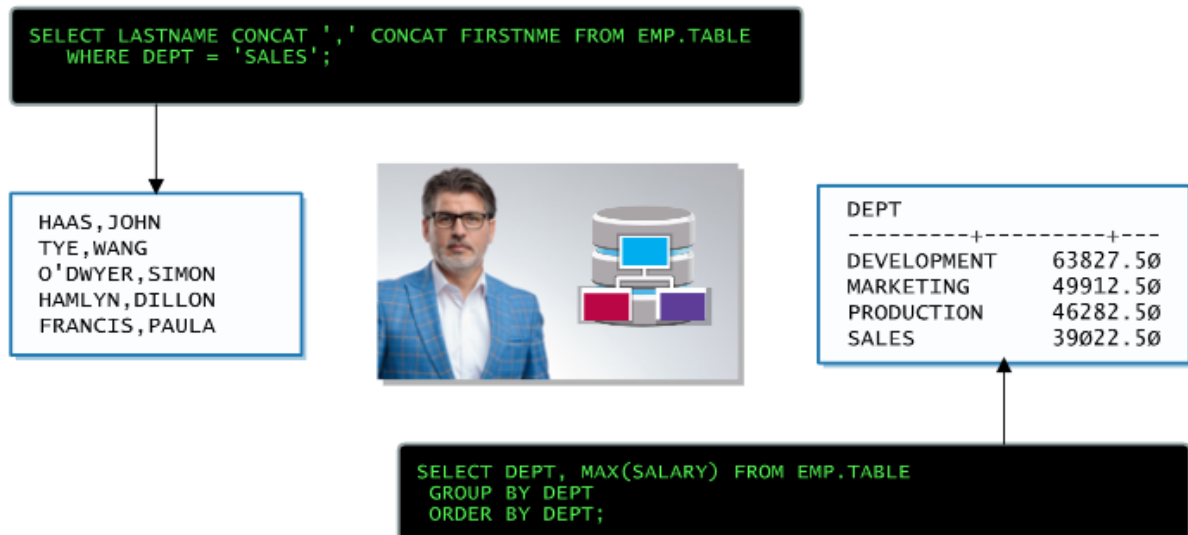   **WHERE BURB LIKE '%SOUTH%';**

- **Scenario 6** - You need to display certain users that have not completed a test.

  SELECT USEDID, SCORE FROM RESULTS.TABLE
     WHERE USERID > 200
     AND SCORE IS NULL
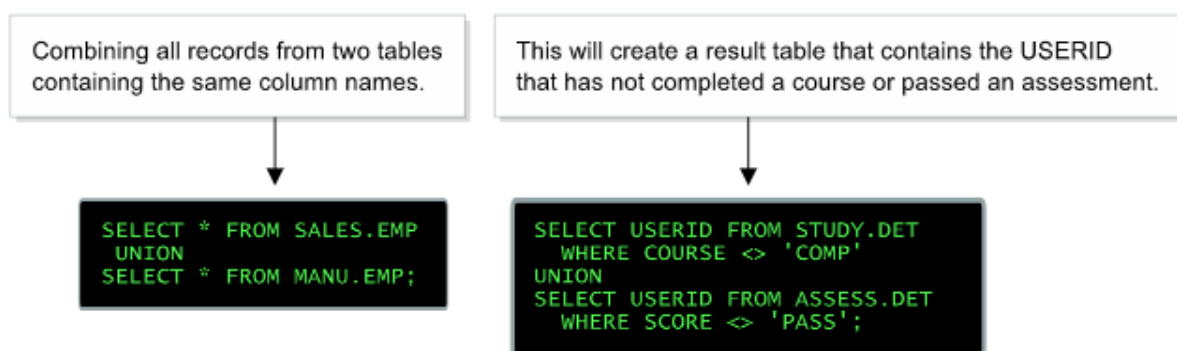   **ORDER BY USERID;**

Functions

There are a number of built-in and user-defined functions that you can also use to further modify the data captured through your SELECT statement.

The CONCAT function shown at the top of this graphic is concatenating character strings, while MAX and MIN can be used to return the maximum and minimum value from a set of values. In this example the MAX function is being used to display the largest salary within each department.

```
SELECT LASTNAME CONCAT ',' CONCAT FIRSTNME FROM EMP.TABLE
    WHERE DEPT = 'SALES';
```

```
HAAS,JOHN
TYE,WANG
O'DWYER,SIMON
HAMLYN,DILLON
FRANCIS,PAULA
```

```
DEPT
---------+---------+---
DEVELOPMENT      63827.5Ø
MARKETING        49912.5Ø
PRODUCTION       46282.5Ø
SALES            39Ø22.5Ø
```

```
SELECT DEPT, MAX(SALARY) FROM EMP.TABLE
 GROUP BY DEPT
 ORDER BY DEPT;
```

Merging Data

The majority of examples shown so far have dealt with a SELECT statement creating data in a result table. The merging of data with that from a separate SELECT statement is possible using the UNION keyword.

Combining all records from two tables containing the same column names.

This will create a result table that contains the USERID that has not completed a course or passed an assessment.

```
SELECT * FROM SALES.EMP
 UNION
SELECT * FROM MANU.EMP;
```

```
SELECT USERID FROM STUDY.DET
   WHERE COURSE <> 'COMP'
UNION
SELECT USERID FROM ASSESS.DET
   WHERE SCORE <> 'PASS';
```

Creating and Modifying Table Data

Creating Tables

The CREATE TABLE statement is used to create the structure into which data is added. A number of data value types can be assigned to columns you define within the table.

```
CREATE TABLE  STAFF.DAT
   (STAFFID        CHAR(4)        NOT NULL,
    FIRSTNAME      VARCHAR(15)    NOT NULL,
    LASTNAME       VARCHAR(25)    NOT NULL,
    SALARY         DECIMAL(8,2),
    DEPTCODE       CHAR(3),
    GENDER         CHAR(1)        NOT NULL,
    PRIMARY KEY    (STAFFID));
```

- In this example, the CREATE TABLE statement is being used to define a new table called STAFF.DAT
- FIRSTNAME and LASTNAME - These are the names of the columns to be created within the table.
- CHAR(4) and VARCHAR(15) - A number of different data types can be assigned to column data. For example, CHAR represents a fixed length character string which is followed by the length, while VARCHAR is used for a variable length character string up to the maximum number of characters specified.
- DECIMAL(8,2) - The DECIMAL data type is used for a column that displays decimal numbers. The first integer is the precision of the number. That is, the total number of digits, which can range from 1 to 31. The second integer is the scale of the number. That is, the number of digits to the right of the decimal point, which can range from 0 to the precision of the number.
- NOT NULL - This is used to prevent the column from containing a null value.
- PRIMARY KEY - This code is used to define a primary key for the table.

Inserting Data

There are several ways of inserting data into a table. You can add a single row of data by specifying values to be inserted into columns, or populate an entire table by inserting all or selected rows from another table.

```
INSERT INTO RESULT.TAB
   VALUES ('002', 'SALES', '109', 'SA', 'UNICORN');
```
This will create a new row within the RESULT.TAB table and add the specified values to the existing columns.

```
INSERT INTO EMP.DAT (DEPTNO, DEPTNAME, MGRNO, SALARY)
   VALUES ('020', 'PRODUCTION', '7099', '52000');
```
This will create a new row in the EMP.DAT table but only add the values to the corresponding column names specified in the first line.

```
INSERT INTO SALES.EMP
   SELECT * FROM MARKET.EMP;
```
This will insert all the data from the MARKET.EMP table into the SALES.EMP table.

```
INSERT INTO ORDER.TAB
   SELECT * FROM OSORDER.TAB
   WHERE LANG = 'ENGLISH';
```
This will only select the records from the OSORDER.TAB table that contain ENGLISH in the LANG column, and then insert them into the ORDER.TAB table.

Deleting Data

Just as rows can be added, they can also be removed using the DELETE statement. A search argument can be provided to remove specific rows or they can be deleted relative to where the cursor is positioned.

- **Example 1** - This will remove all data from the TSCORES.DAT table.
- **Example 2** - This statement is used to remove a specific record from the table, that being the record for employee 022.
- **Example 3** - This statement is used to remove multiple records from the ACTIVITY.TAB table, those being where the DEPTNAME column contains the value MARKETING.
- **Example 4** - In this example, a previous statement has declared C1 as the cursor. This DELETE statement will delete the row on which the cursor C1 is currently positioned.

```
DELETE FROM TSCORES.DAT;
```

```
DELETE FROM ORG.EMP
 WHERE EMPNO = '022';
```

```
DELETE FROM ACTIVITY.TAB
 WHERE DEPTNAME = 'MARKETING';
```

```
DELETE FROM PLANT.DET
  WHERE CURRENT OF C1;
```

Updating Existing Data

When existing table data needs to be modified, the UPDATE statement can be used. This statement can update data on one or multiple rows and can be used in conjunction with search criteria.

```
UPDATE USORG.EMP
  SET MANAGER = 'YOUNG'
  WHERE DEPTNO = '005';
```

This will update only those records in table USORG.EMP where the DEPTNO column contains the 005 value. In those records, the value in the MANAGER column will be changed to YOUNG.

```
UPDATE TAX.EMP
  SET SALARY = 1.05 * SALARY;
```

This will update all records in table TAX.EMP, changing the value in the SALARY column to reflect a 5% increase.

SQL and SPUFI

Accessing DB2 Data - SPUFI

As one begins coding SQL you will need to test it against DB2 data. A quick method of performing this is to use the SQL Processor Using File Input (SPUFI) facility, which is available through TSO/ISPF.

Accessing SPUFI

Because SPUFI is run through TSO/ISPF, you will need to have been assigned access to this product by your security administrator who will provide you with a logon ID and password. After logging on, an ISPF Primary Option panel similar to the one shown here will be displayed.

```
 Menu  Utilities  Compilers  Options  Status  Help
---------------------------------------------------------------------------
                        ISPF Primary Option Menu
 Option ===>
                                      More:       +
 0  Settings      Terminal and user parameters        User ID . : IBMUSER
 1  View          Display source data or listings      Time. . . : 14:54
 2  Edit          Create or change source data         Terminal. : 3278
 3  Utilities     Perform utility functions            Screen. . : 1
 4  Foreground    Interactive language processing      Language. : ENGLISH
 5  Batch         Submit job for language processing   Appl ID . : ISR
 6  Command       Enter TSO or Workstation commands    TSO logon : DBPROCCG
 7  Dialog Test   Perform dialog testing               TSO prefix: GHPROD
 8  LM Facility   Library administrator functions      System ID : S0w1
 9  IBM Products  IBM program development products     MVS acct. : FB3
 10 SCLM          SW Configuration Library Manager     Release . : ISPF 7.4
 11 Workplace     ISPF Object/Action Workplace

                  ------ Other Install Products ------

 MQ MQSeries      MQSseries 5.3.1
 D2 DB2I          Perform DB2 Interactive functions
```

In this example you can see option D2 at the bottom of the screen, which is used to access the SPUFI facility on this system. Note that your site's systems programmer may use other panel option selection codes, or that you may need to scroll down (PF8) in your version of this panel to locate the correct option

```
                        DB2I PRIMARY OPTION MENU          SSID: DBCG
 COMMAND ===>

 Select one of the following DB2 functions and press ENTER.

 1   SPUFI                  (Process SQL statements)
 2   DCLGEN                 (Generate SQL and source language declarations)
 3   PROGRAM PREPARATION    (Prepare a DB2 application program to run)
 4   PRECOMPILE             (Invoke DB2 precompiler)
 5   BIND/REBIND/FREE       (BIND, REBIND, or FREE plans or packages)
 6   RUN                    (RUN an SQL program)
 7   DB2 COMMANDS           (Issue DB2 commands)
 8   UTILITIES              (Invoke DB2 utilities)
 D   DB2I DEFAULTS          (Set global parameters)
 X   EXIT                   (Leave DB2I)




 PRESS:                    END to exit     HELP for more information
```

You can see that a number of interactive DB2 functions can be accessed from the resulting DB2I Primary Option Menu. The DB2 subsystem which will perform those functions is identified in the top right of this, and subsequent menu panels. In this case, it is DB11.
- The identity of the subsystem is just one of the variables which may be set via the panel option: D DB2I DEFAULTS.

SPUFI Panel - Overview

The SPUFI panel allows you to provide details on the input, output, and general SPUFI defaults to be used when invoking your SQL statements against DB2 table data and indexes.
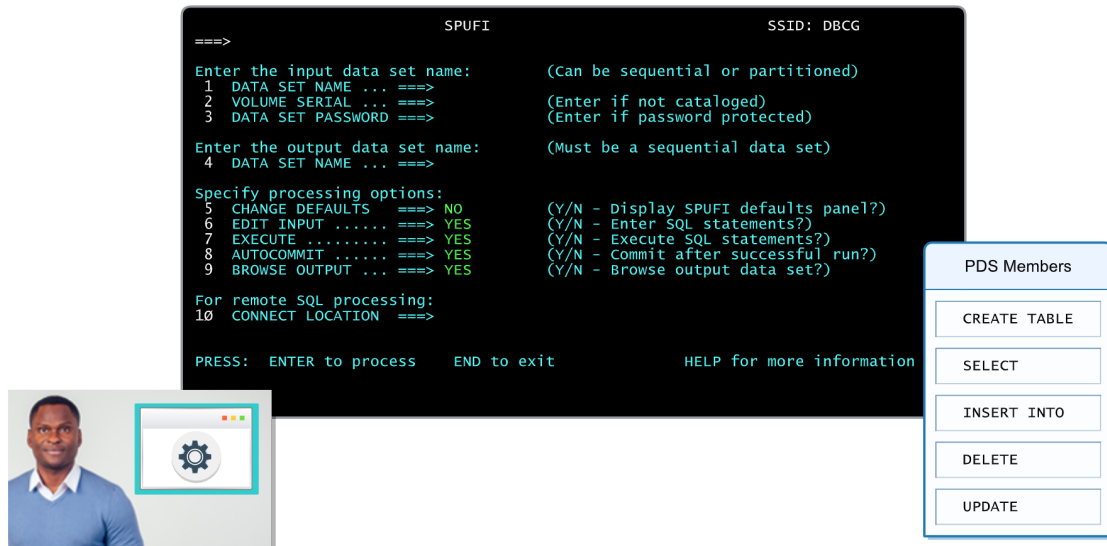
Firstly, you should create a data set to store the SQL statements you need to test. This can be a sequential data set, but if you need to test different types of statements you may want to create a partitioned data set so that unique code can be stored in different members.

SPUFI Panel - Input
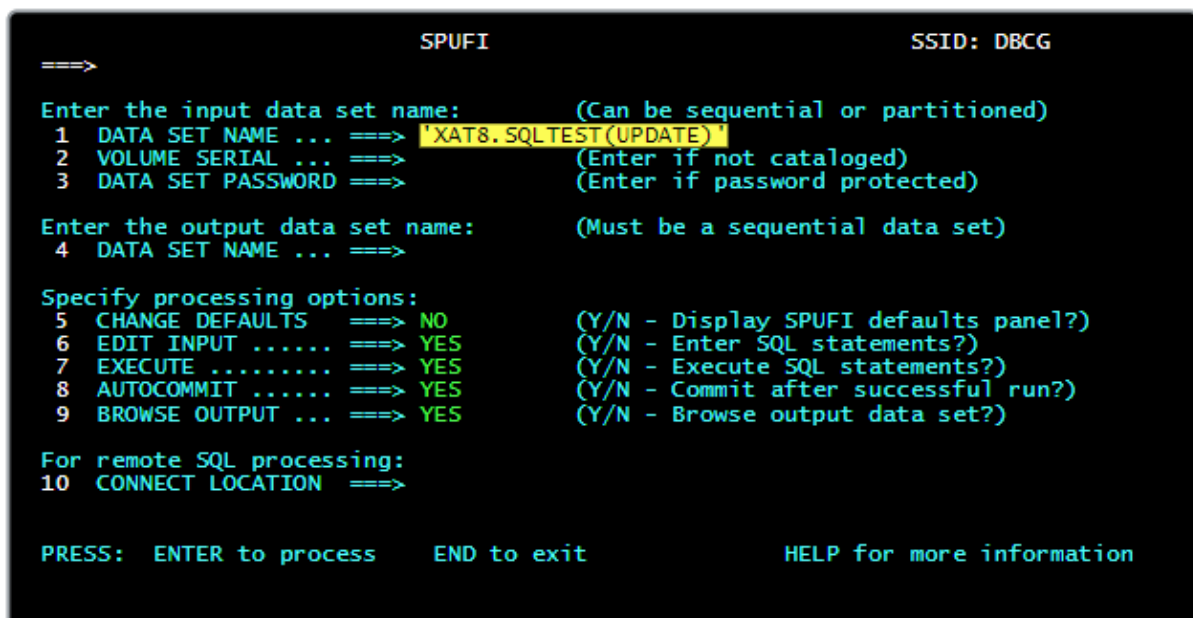
You will enter some SQL statements later but for now, you will code your SQL statements into a data set called XAT8.SQLTEST in a member called UPDATE.

This data set name and member needs to be coded within single quotes in the input data set name field. If the name is spelled incorrectly or you forget to include the single quotes, an error message will be displayed when SPUFI attempts to execute the SQL.

SPUFI Panel - Results

The result from your executed SQL statements will be stored in a sequential data set specified in option 4. If the data set name you provide is an existing one, data in it will be overwritten each time you use it. If the data set name does not exist, it will be created automatically for you.

```
                        SPUFI                      SSID: DBCG
===>

Enter the input data set name:        (Can be sequential or partitioned)
 1  DATA SET NAME ... ===> 'XAT8.SQLTEST(UPDATE)'
 2  VOLUME SERIAL ... ===>              (Enter if not cataloged)
 3  DATA SET PASSWORD ===>              (Enter if password protected)

Enter the output data set name:       (Must┌─────────────────────────────┐
 4  DATA SET NAME ... ===> RESULT  ◄───  │ If your logon ID is XAT8 then the data set │
                                         │ referenced here would be XAT8.RESULT       │
Specify processing options:             └─────────────────────────────┘
 5  CHANGE DEFAULTS   ===> NO        (Y/N - Display SPUFI defaults panel?)
 6  EDIT INPUT ...... ===> YES       (Y/N - Enter SQL statements?)
 7  EXECUTE ......... ===> YES       (Y/N - Execute SQL statements?)
 8  AUTOCOMMIT ...... ===> YES       (Y/N - Commit after successful run?)
 9  BROWSE OUTPUT ... ===> YES       (Y/N - Browse output data set?)

For remote SQL processing:
10  CONNECT LOCATION  ===>


PRESS:  ENTER to process    END to exit        HELP for more information
```

If you forget to insert the single quotes around the data set name, the data set referenced will contain your logon ID as the high-level qualifier.

SPUFI Defaults

There may be cases where the default processing values used by SPUFI need to be changed before your SQL statements are processed. By setting the value for option 5 to Y or YES, the SPUFI default panel will be displayed before executing your SQL statements.

```
                        SPUFI                      SSID: DBCG
==>

Enter the input data set name:        (Can be sequential or partitioned)
 1  DATA SET NAME ... ===> 'XAT8.SQLTEST(UPDATE)'
 2  VOLUME SERIAL ... ===>              (Enter if not cataloged)
 3  DATA SET PASSWORD ===>              (Enter if password protected)

Enter the output data set name:       (Must be a sequential data set)
 4  DATA SET NAME ... ===> RESULT

Specify processing options:
 5  CHANGE DEFAULTS   ===> NO         (Y/N - Display SPUFI defaults panel?)
 6  EDIT INPUT ...... ===> YES        (Y/N - Enter SQL statements?)
 7  EXECUTE ......... ===> YES        (Y/N - Execute SQL statements?)
 8  AUTOCOMMIT ...... ===> YES        (Y/N - Commit after successful run?)
 9  BROWSE OUTPUT ... ===> YES        (Y/N - Browse output data set?)

For remote SQL processing:
10  CONNECT LOCATION  ===>


PRESS:  ENTER to process    END to exit        HELP for more information
```

The pages that follow contain details on the defaults you can overtype and update on the CURRENT SPUFI DEFAULTS panel.

- **SQL Terminator** - As mentioned in the sections above, the default SQL statement terminator is a semicolon. If your SQL statement does not contain this character as part of its data processing, then SPUFI will not know when to terminate the statement.
- **Isolation Level** - SQL Statements can be executed at various isolation levels to prevent them from affecting other applications that are running.
- **Max Select Lines** - If you are using your SQL SELECT statements for testing purposes, you may want to limit the amount of data that is returned from it. The MAX SELECT LINES field can contain a number from 1 to 99999999 and will prevent any more lines than the number specified from being selected.
- **Allow SQL Warnings** - This field is used to specify the action to take should an SQL warning occur. If YES is specified, then SPUFI is allowed to continue processing the SQL statement. If NO is specified, then SPUFI will stop processing the SQL statement.
- **Change Plan Names** - Specifying YES will display another panel that allows you to change the plan name to be used by SPUFI
- **SQL Format** - This option is used to instruct SPUFI on how to pre-process SQL statements. The SQL value is the most commonly specified option except when using SQL procedural language. The SQLCOMNT and SQLPL values are often used for SQL procedural language processing.

If the output produced from your SQL is to be used for other purposes such as being fed as input into another program, you may need to change some of the output data set characteristics or output format characteristics. Normally once all defaults have been defined in this panel, you will not need to access it every time you process your SQL, so the CHANGE DEFAULTS option from the previous panel can be set to NO.

The F3 key can be used to exit from this panel and return to the main SPUFI panel, or if you have made any required changes, the Enter key can be pressed to continue processing of your SQL statements.

SPUFI Panel - Editing Input

In this scenario there were no changes made to the defaults, and a message informing you of that fact is displayed at the top of the SPUFI panel.

```
                        SPUFI                        SSID: DBCG
 ===>
 DSNE800A NO DEFAULT VALUES WERE CHANGED. PRESS ENTER TO CONTINUE
 Enter the input data set name:       (Can be sequential or partitioned)
  1  DATA SET NAME ... ===> 'XAT8.SQLTEST(UPDATE)'
  2  VOLUME SERIAL ... ===>           (Enter if not cataloged)
  3  DATA SET PASSWORD ===>           (Enter if password protected)

 Enter the output data set name:      (Must be a sequential data set)
  4  DATA SET NAME ... ===> RESULT

 Specify processing options:
  5  CHANGE DEFAULTS  ===> *          (Y/N - Display SPUFI defaults panel?)
  6  EDIT INPUT ...... ===> YES       (Y/N - Enter SQL statements?)
  7  EXECUTE ......... ===> YES       (Y/N - Execute SQL statements?)
  8  AUTOCOMMIT ...... ===> YES       (Y/N - Commit after successful run?)
  9  BROWSE OUTPUT ... ===> YES       (Y/N - Browse output data set?)

 For remote SQL processing:
 10  CONNECT LOCATION  ===>


 PRESS:  ENTER to process    END to exit            HELP for more information
```

The CHANGE DEFAULTS field is also changed to * to indicate that it has been processed and that when the Enter key is pressed the defaults panel will not be displayed.

There are two methods of entering your SQL statements through SPUFI:
- You can create your input data set or data set member and using ISPF, code all your required statements prior to invoking them through SPUFI.
- Another method is to type YES into the EDIT INPUT field so that when SPUFI is invoked, the input data set will be opened using ISPF Edit and you can type in the statements required for that invocation.

The SQL statements required can be entered in the data area of this screen, either into an empty line or by overtyping existing text. When all statements have been entered, the PF3 key or END command can be used to save the data and return to the SPUFI panel.

SPUFI Panel - Processing

The message at the top of the SPUFI panel indicates that the editing has been performed on the input data set. Normally you will want to begin processing the SQL statements that you have entered in your input data set, so the EXECUTE field should be set to Y or YES before pressing the Enter key.

```
                              SPUFI                       SSID: DBCG
 ===>
 DSNE808A EDIT SESSION HAS COMPLETED. PRESS ENTER TO CONTINUE
 Enter the input data set name:          (Can be sequential or partitioned)
  1   DATA SET NAME ... ===> 'XAT8.SQLTEST(UPDATE)'
  2   VOLUME SERIAL ... ===>              (Enter if not cataloged)
  3   DATA SET PASSWORD ===>              (Enter if password protected)

 Enter the output data set name:         (Must be a sequential data set)
  4   DATA SET NAME ... ===> RESULT

 Specify processing options:
  5   CHANGE DEFAULTS   ===> *            (Y/N - Display SPUFI defaults panel?)
  6   EDIT INPUT ...... ===> *            (Y/N - Enter SQL statements?)
  7   EXECUTE ......... ===> YES          (Y/N - Execute SQL statements?)
  8   AUTOCOMMIT ...... ===> YES          (Y/N - Commit after successful run?)
  9   BROWSE OUTPUT ... ===> YES          (Y/N - Browse output data set?)

 For remote SQL processing:
 10   CONNECT LOCATION  ===>


 PRESS:  ENTER to process     END to exit            HELP for more information
```

If this field is set to NO and the BROWSE OUTPUT field is set to YES, then no SQL processing will be performed but any existing data in the output data set specified will be displayed.

SPUFI Panel - Interrupting Processing

If the EXECUTE field is set to YES in your SPUFI panel, the next stage when you press Enter will see the execution of your SQL statements.

If there are many statements in your input data set or if the table you are interrogating is large, it may take considerable time to process the statements. If this is unacceptable, you can interrupt the processing by using the PA1 key, with the amount of output created depending on how much processing the statement(s) were able to perform before being interrupted.

If SPUFI terminates while executing your statements, it could indicate that DB2 resource limits have been defined by your DB2 administrator and that your SQL execution has exceeded them.

Note: The AUTOCOMMIT option on this page has been changed to NO, so that you can see the SPUFI COMMIT OR ROLLBACK panel on the next page.

```
 Specify processing options:
  5   CHANGE DEFAULTS   ===> *            (Y/N - Display SPUFI defaults panel?)
  6   EDIT INPUT ...... ===> *            (Y/N - Enter SQL statements?)
  7   EXECUTE ......... ===> YES          (Y/N - Execute SQL statements?)
  8   AUTOCOMMIT ...... ===> NO           (Y/N - Commit after successful run?)
  9   BROWSE OUTPUT ... ===> YES          (Y/N - Browse output data set?)
```

SPUFI Panel - Saving Changes

Normally when processing SQL statements you will want the results saved to the Db2 tables if they have executed successfully. If you are testing statements then you can be given the option to commit them to the database by specifying N or NO in the AUTOCOMMIT field on the SPUFI panel. This will result in the screen shown here being displayed.

```
                              SPUFI COMMIT OR ROLLBACK          SSID: DBCG
===>
DSNE361I SPUFI PROCESSING COMPLETE
 Explanation:

          The last SQL statement ran with a return code of -204
          Because AUTOCOMMIT was set to NO, you must now decide whether to:

          * COMMIT (save) your database changes or to
          * ROLLBACK (erase) your database changes or to
          * DEFER the decision, which will allow you to continue processing
            another input data set. If you select DEFER, it is possible
            that you will tie up system resources that others need.

 1  NEXT ACTION ===> COMMIT    (Enter COMMIT, ROLLBACK, or DEFER)




 PRESS:  ENTER to process                       HELP for more information
```

SPUFI Panel - Remote Servers

If your SQL statements need to be processed on a remote database server, then the location of that server can be entered into the CONNECT LOCATION field.

```
 9  BROWSE OUTPUT ... ===> YES      (Y/N - Browse output data set?)

For remote SQL processing:
10  CONNECT LOCATION  ===> TESTDB

PRESS:  ENTER to process     END to exit
```

If SPUFI is unable to connect to your specified CONNECT LOCATION, then the output data set will display return codes and error messages relating to the attempt.

Interpreting SPUFI Output

Output Format

The ISPF Browse facility is opened and the result of your SQL execution is displayed. At the top of the output produced, the following data is displayed:
- A copy of the SQL statements from the input data set
- The data produced as a result of the processed SQL statements

As you scroll toward the bottom of the data, you may encounter warning or error messages associated with the invocation, and will also see summary statistics.

Sample 1 - Error Message

The example shown here highlights a message indicating that there is a problem. The message says that the table name specified in the SELECT statement is undefined, which indicates that a table of that name could not be located.

As a result of this error, rollback processing is performed.

```
 Menu  Utilities  Compilers  Help
 --------------------------------------------------------------------
 BROWSE     XAT8.RESULT                         Line 00000000 Col 001 080
 Command ===>                                        Scroll ===> CSR
 *************************** Top of Data ******************************
 ---------+---------+---------+---------+---------+---------+---------+
 SELECT * FROM CHIG.EMP;                                       00010012
 ---------+---------+---------+---------+---------+---------+---------+
 DSNT408I SQLCODE = -204, ERROR: CHIG.EMP IS AN UNDEFINED NAME
 DSNT418I SQLSTATE   = 42704 SQLSTATE RETURN CODE
 DSNT415I SQLERRP    = DSNXOTL SQL PROCEDURE DETECTING ERROR
 DSNT416I SQLERRD    = -500 0   0  -1   0   0 SQL DIAGNOSTIC INFORMATION
 DSNT416I SQLERRD    = X'FFFFFE0C' X'00000000' X'00000000' X'FFFFFFFF'
          X'00000000' X'00000000' SQL DIAGNOSTIC INFORMATION
 ---------+---------+---------+---------+---------+---------+---
 DSNE618I ROLLBACK PERFORMED, SQLCODE IS 0  ◄───
 DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
 ---------+---------+---------+---------+---------+---------+---
 DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
 DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
 DSNE621I NUMBER OF INPUT RECORDS READ IS 1
 DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 17
 *************************** Bottom of Data ***************************
```

This message occurs when an error is encountered and the AUTOCOMMIT field on the SPUFI panel is set to YES. Rollback will undo any data changes made since the last commit.

Sample 2 - Error Message

In this example, the messages indicate that there is an illegal symbol being used to terminate one of the SQL statements. In fact, the problem is associated with a missing corresponding closed parenthesis on the last line.

This highlights that at times it can be difficult to determine the actual cause of a problem from the messages displayed.

```
 Menu  Utilities  Compilers  Help
 --------------------------------------------------------------------
 BROWSE     XAT8.RESULT                         Line 00000000 Col 001 080
 Command ===>                                        Scroll ===> CSR
 *************************** Top of Data ******************************
 ---------+---------+---------+---------+---------+---------+---------+
 CREATE TABLE COURSE5.DET                                      00010001
    (EMPNO CHAR(6) NOT NULL,                                   00020001
     COURSENME VARCHAR(36) NOT NULL,                           00030001
     SCORE CHAR(6),                                            00040001
     ADMRDEPT CHAR(3) NOT NULL,                                00050001
     LOCATION CHAR(16),                                        00060001
     PRIMARY KEY(EMPNO);                                       00070001
 ---------+---------+---------+---------+---------+---------+---------+
 DSNT408I SQLCODE = 104, ERROR:  ILLEGAL SYMBOL "<END-OF-STATEMENT>". SOME
          SYMBOLS THAT MIGHT BE LEGAL ARE: , )
 DSNT418I SQLSTATE   = 42601 SQLSTATE RETURN CODE
 DSNT415I SQLERRP    = DSNHPARS SQL PROCEDURE DETECTING ERROR
 DSNT416I SQLERRD    = 3 0   0  -1   452   502 SQL DIAGNOSTIC INFORMATION
 DSNT416I SQLERRD    = X'00000003' X'00000000' X'00000000' X'FFFFFFFF'
          X'000001C4' X'000001F6' SQL DIAGNOSTIC INFORMATION
 ---------+---------+---------+---------+---------+---------+---------+
 DSNE618I ROLLBACK PERFORMED, SQLCODE IS 0
 DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
```

Batch SPUFI

There is a batch equivalent that you may prefer to use if you have a large amount of SQL statements you need to invoke or there is a big Db2 table you need to access. Several batch programs are available that perform functions similar to those found in SPUFI, with DSNTEP2 and DSNTEP4 being the most popular.

An example of batch SPUFI is shown here.

```
   File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
----------------------------------------------------------------------------
EDIT       JOHN.SPUFI.PDS(BATSPUF1) - 01.00            Columns 00001 00072
Command ===>                                            Scroll ===> CSR
****** *************************** Top of Data ******************************
000001 //BATSPUF1  JOB (20,FB3),JOHN,MSGLEVEL=(1,1),TIME=20,
000002 //             CLASS=A,MSGCLASS=H,NOTIFY=JOHN,REGION=0M
000003 //DSNC10     JCLLIB ORDER=(VENDOR.PROCLIB,DSNC10.PROCLIB)
000004 //JOBLIB     DD DISP=SHR,DSN=DSNC10.SDSNLOAD
000005 //TEST1AB    EXEC PGM=IKJEFT01,DYNAMNBR=20
000006 //SYSTSPRT   DD   SYSOUT=*
000007 //SYSPRINT   DD   SYSOUT=*
000008 //SYSTSIN    DD   *
000009   DSN SYSTEM(DBCG)
000010   RUN PROGRAM(DSNTEP2)  PLAN(DSNTEP12) -
000011       LIB('DSNC10.DBCG.RUNLIB.LOAD') -
000012       PARMS('/ALIGN(LHS) MIXED TOLWARN(YES)')
000013   END
000014 //SYSIN     DD   *
000015    SELECT * FROM DSN81210.EMP;
000016 /*
****** *************************** Bottom of Data ***************************
```