RDBMS Basics, IBM DB2 SQL and SQL/PL

Preface

This document will take you through RDBMS Basics, IBM DB2 SQL and SQL/PL.

Table of contents | **Page No**

1. Introduction To DBMS

1.1 Data

Activities that are need to be recorded known as Data. By relating different pieces of data, we are able to extract valuable **information** by presenting it in meaningful context.

1.2 Database

A **database** is a collection of information that is organized so that it can easily be accessed, managed and updated. It must provide proper storage for large amounts of data, easy and fast access and facilitate the processing of data.

1.3 Importance of Database

Organizations must track information about people including volunteers, clients, potential donors, current donors, event attendees etc. Managing this information is crucial.

A database allows you to manage and use an incredible variety of information easily.

1.4 DBMS (DataBase Management System)

DBMS is a set of software that is used to define, store, manipulate and control the data in a database.

1.5 RDBMS (Relational DataBase Management System)

A **RDBMS** is a **DBMS** in which data is stored in tables and the relationships among the data are also stored in tables. The data can be accessed or reassembled in many different ways without having to change the table forms.

RDBMS is based on the relational model as introduced by E. F. Codd. Most popular databases currently in use are based on the relational database model.

RDBMS contains a set of objects used to store, manage, and access data. Examples of such objects are tables, views, indexes, functions, triggers, and packages.
In RDBMS

1.5.1 Popular RDBMS Vendors

S.No	Product Name	Vendor
1	DB2	IBM
2	Oracle	Oracle Corporation
3	SQL Server	Microsoft Corporation

4	MySQL	Sun Microsystems – Now: Oracle Corporation
5	Teradata	NCR

1.6 Table, Column, Row, Value

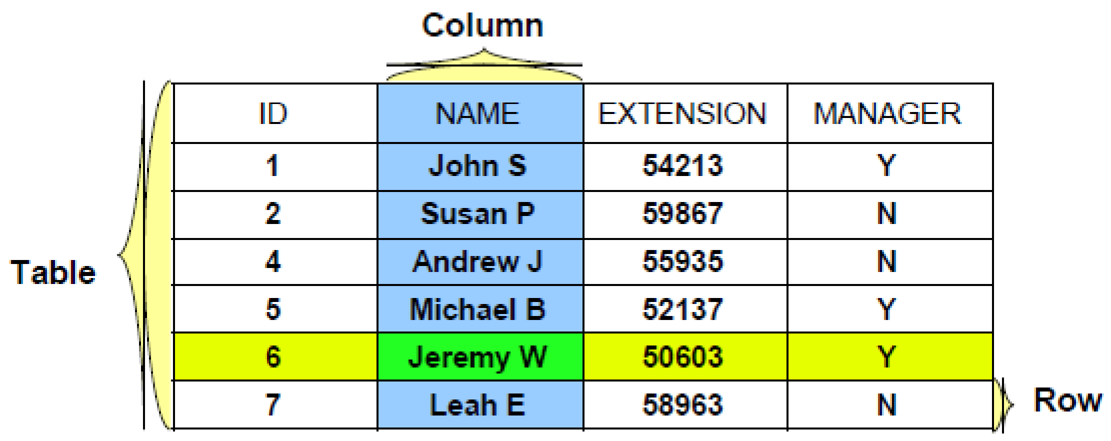
Table RDBMS stores data in the form of Tables. A table is an unordered set of records, consisting of rows and columns.

Column Column is a set of values of the same type, so each column has a specific data type. Logically columns are called as Attributes.

Row Each row features a certain value for each column or each row represents an entry in the table. A row is a sequence of values such that the nth value is a value of the nth column of the table. Logically rows are called as Entities.

Value At the intersection of every column and row is a specific data item called a value. Logically Values are called as Tuples.

Note: Every table must have one or more columns, but the number of rows can be zero.



ID	NAME	EXTENSION	MANAGER
1	John S	54213	Y
2	Susan P	59867	N
4	Andrew J	55935	N
5	Michael B	52137	Y
6	Jeremy W	50603	Y
7	Leah E	58963	N

1.7 Normalization

The process of organizing data to minimize redundancy in the design of a relational database management system (RDBMS) is called normalization. The goal of database normalization is to decompose relations with anomalies in order to produce smaller, well-structured relations. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them.

Edgar F. Codd, the inventor of the relational model, introduced the concept of normalization and what we now know as the First Normal Form (1NF) in 1970. Codd went on to define the Second Normal Form (2NF) and Third Normal Form (3NF) in 1971, and Codd and Raymond F. Boyce defined the Boyce-Codd Normal Form (BCNF) in 1974. Higher normal forms were defined by other theorists in subsequent years, the most recent being the Sixth Normal Form (6NF) introduced by

Chris Date, Hugh Darwen, and Nikos Lorentzos in 2002.

Informally, a relational database table (the computerized representation of a relation) is often described as "normalized" if it is in the Third Normal Form. Most 3NF tables are free of insertion, update, and deletion anomalies, i.e. in most cases 3NF tables adhere to BCNF, 4NF, and 5NF (but typically not 6NF).

2. Introduction to IBM DB2

IBM DB2 is a relational model database server developed by IBM. There are three DB2 products that are very similar, but not identical: DB2 for LUW (Linux, Unix, and Windows), DB2 for z/OS (mainframe), and DB2 for iSeries (formerly OS/400). The DB2 LUW product runs on multiple Linux and UNIX distributions, such as Red Hat Linux, SUSE Linux, AIX, HP/UX, and Solaris, and most Windows systems.

2.1 Brief History of DB2

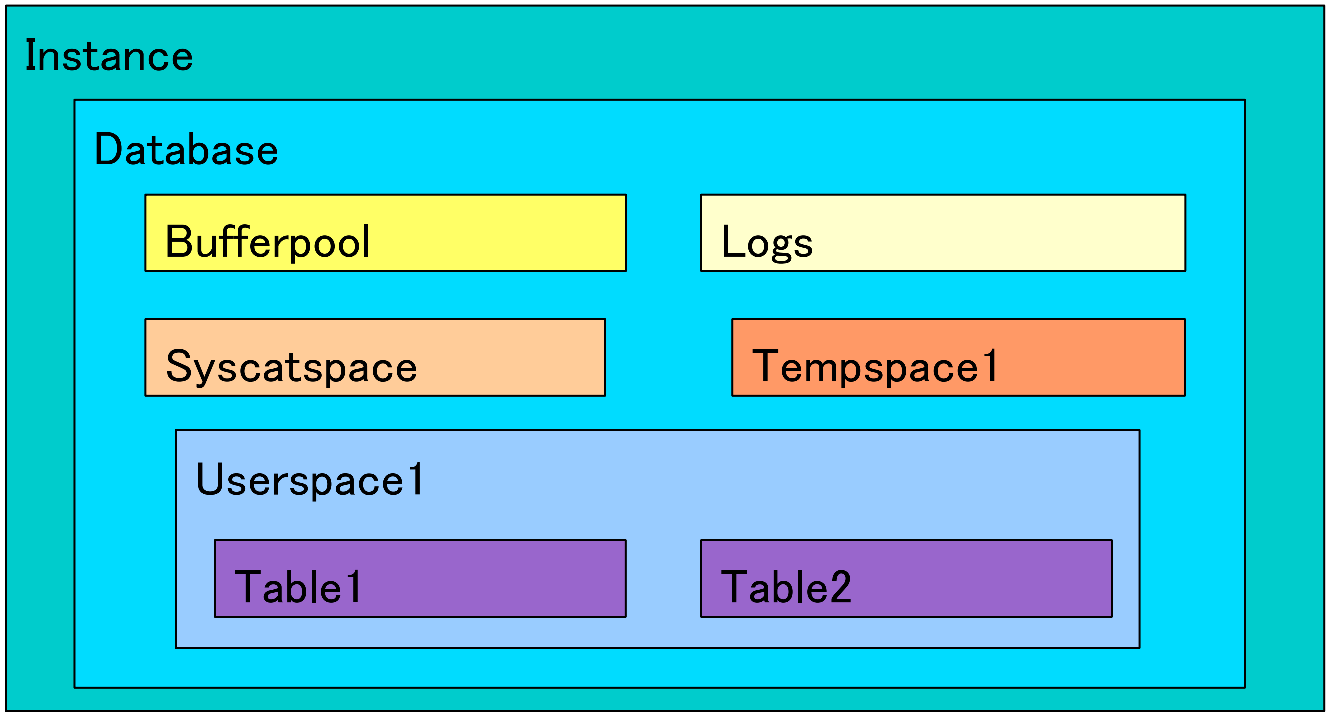
Since the 1970s, when IBM Research invented the Relational Model and the Structured Query Language (SQL), IBM has developed a complete family of data servers. Development started on mainframe platforms such as Virtual Machine (VM), Virtual Storage Extended (VSE), and Multiple Virtual Storage (MVS). In 1983, DB2 for MVS Version 1 was born. “DB2” was used to indicate a shift from hierarchical databases—such as the Information Management System (IMS) popular at the time—to the new relational databases. DB2 development continued on mainframe platforms as well as on distributed platforms.

In 1996, IBM announced DB2 *UDB* Version 5 for distributed platforms. With this version, DB2 was able to store all kinds of electronic data, including traditional relational data, as well as audio, video, and text documents. It was the first version optimized for the Web, and it supported a range of distributed platforms;for example, OS/2, Windows, AIX, HP-UX, and Solaris;from multiple vendors.

2.2 IBM DB2 Versions

- In 1996, IBM announced DB2 *UDB* Version 5 for distributed platforms
- In the mid 2000's IBM released DB2 V8 UDB
- In mid 2006, IBM announced DB2 V9.1
- In October 2007, IBM announced DB2 V9.5
- In June 2009, IBM announced DB2 V9.7
- In May 2012, IBM released DB2 V10.1
- In June 2013, IBM released DB2 V10.5

2.3 DB2 Architecture



2.3.1 Instance

In DB2, an instance provides an independent environment where databases can be created and applications can be run against them. Because of these independent environments, databases in separate instances can have the same name.

2.3.2 Databases

A database is a collection of information organized into interrelated objects like table spaces, tables, and indexes. Databases are closed and independent units associated to an instance. Because of this independence, objects in two or more databases can have the same name.

2.3.3 Tablespaces

Table spaces are logical objects used as a layer between logical tables and physical containers. Containers are where the data is physically stored in files, directories, or raw devices. When you create a table space, you can associate it to a specific buffer pool (database cache) and to specific containers.

Three table spaces SYSCATSPACE, TEMPSPACE1 and USERSPACE1 are automatically created when you create a database.

2.3.4 Bufferpools

A Bufferpool is an area in memory where all index and data pages other than LOBs are processed. DB2 retrieves LOBs directly from disk.

2.3.5 Logs

Logs are used by DB2 to record every operation against a database. In case of a failure, logs are crucial to recover the database to a consistent point.

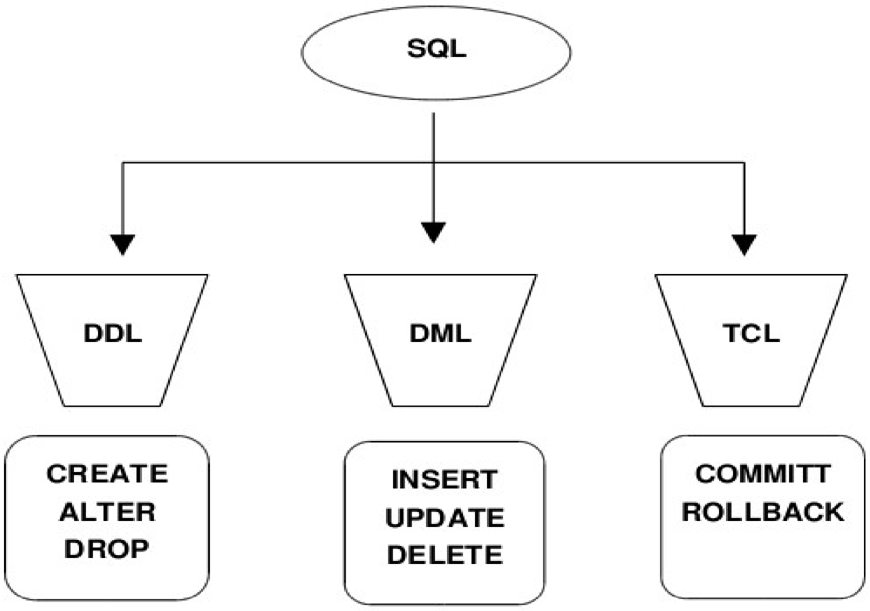
2.4 Competition

IDC's Worldwide Database Management Systems 2009–2013 Forecast and 2008 Vendor Shares[2] ranks Oracle database as the leader in DBMS marketing share, followed by IBM DB2 and then Microsoft SQL Server. Other competitors include open source products such as Firebird, PostgreSQL, MySQL and Ingres, and niche players such as Sybase and MaxDB.

In 2009, Gartner declared that "IBM DB2 9.7 Shakes Up the DBMS Market With Oracle Compatibility".[3] This headline refers to the addition to DB2 of several features that are familiar to users of Oracle Database, making it easier for people with Oracle Database skills to work with DB2. These new features include DB2 support for the most commonly used SQL, PL/SQL, and scripting syntax from Oracle Database. They also include DB2 support for additional data types and concurrency models.

3. Structured Query Language (SQL)

This is a common language with which we can interact with the database.
SQL mainly divided into three categories.



3.1 Data types in DB2

Data types	
integer	char
numeric	varchar
float	time
decimal	date
	timestamp

3.2 Constraints

Constraint restricts the values that the table can store.
We can declare integrity constraints at the table level or column level.
There are 5 types of Constraints

3.2.1 Not Null

The NOT NULL constraint enforces a column to NOT accept NULL values.

3.2.2 Unique key

The UNIQUE constraint uniquely identifies each record in a database table. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns. To enforce unique constraint, the column must be a Not Null column.

3.2.3 Check

The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column. If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

3.2.4 Primary key

The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain unique values. A primary key column cannot contain NULL values.

3.2.5 Foreign key

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

3.3 Data Definition language

These commands used to create, modify and delete Database objects.

3.3.1 Create

The Create statement is used to create new table.

Syntax:
```
CREATE TABLE TABLENAME(<COLUMN_NAME1>  <DATATYPE>[<WIDTH>] [CONSTRAINT
<CONSTRAINT NAME> <CONSTRAINT TYPE>]);
```

Example 1
```
create table emp(emp_no integer not null primary key,emp_name varchar(10),job
varchar(9),mgr_id integer,hiredate date,salary decimal(10,2));
```

Example 2
```
create table employee(emp_no integer not null,emp_code integer not
null,first_name varchar(10),last_name varchar(10),state varchar(10),primary
key(emp_no,emp_code));
```

3.3.2 Alter

The Alter statement is used to add new column(s) to existing table.

Syntax to add a column in a table:
```
ALTER TABLE TABLENAME ADD COLUMNNAME DATATYPE
```

Syntax to delete a column in a table:
```
ALTER TABLE TABLENAME DROP COLUMN  COLUMNNAME
```

Syntax to change the data type of a column in a table:
```
ALTER TABLE TABLENAME ALTER COLUMN COLUMNNAME DATATYPE
```

Example 1
```
alter table employee add middle_name varchar(20);
```

Example 2
```
alter table employee drop column middle_name ;
```

Example 3
```
alter table employee alter state set data type char(20);
```

3.3.3 Drop

The Drop statement is used to delete entire table.
Syntax:
```
DROP TABLE TABLENAME
```

Example
```
drop table employee
```

3.4 Data Manipulation Language

These commands are used to append, change or remove the data in a Table.

3.4.1 Insert

It is possible to write the INSERT INTO statement in two forms.

Syntax 1:
```
INSERT INTO TABLENAME VALUES (value1, value2, value3)
```

Syntax 2:
```
INSERT INTO TABLENAME(column1, column2, column3) VALUES (value1, value2, value3)
```

Example 1
```
INSERT INTO EMP VALUES (7369,'SMITH' , 'CLERK', 7566,'12/17/1990', 8212.00,0.00,
20);
```

Example 2
```
insert into emp(emp_no,emp_name) values(7369,'SMITH');
```

3.4.2 Update

The Update statement is used to update existing records in a table.

Syntax 1:
```
UPDATE TABLENAME SET COLUMNNAME=value
```

Syntax 2:
```
UPDATE TABLENAME SET COLUMNNAME=value WHERE CONDITION
```

Example 1
```
update emp set salary=150000;
```

Example 2
```
update emp set dept_no=20 where job='analyst';
```

3.4.3 Delete

The Delete statement is used to delete rows in a table.

Syntax:
```
DELETE FROM TABLENAME WHERE CONDITION
```

Example
```
delete from emp where emp_no=143;
```

3.5 Transaction Control Language

These commands are used to maintain the consistency of the database.

3.5.1 Commit

This command is used to make the transaction permanent.

3.5.2 Rollback

This command is used to undo the recent transaction.

3.6 Operators

3.6.1 Mathematical Operators

Operator Name	Symbolic Notation
Addition	+
Subtraction	-
Division	/
Multiplication	*

3.6.2 COMPARISION OPERATORS

Operator Name	Symbolic Notation
Equal To	=
Greater Than	>
Less Than	<
Greater Than or Equal To	>=
Less Than or Equal To	<=
Not Equal To	<> or !=

3.6.3 Logical Operators

Operator Name
AND
OR
NOT

3.6.4 Special Operators

Operator Name
IN
LIKE
NULL
BETWEEN
HAVING
EXISTS
DISTINCT
ORDER BY

Examples

1.
```
select deptno,deptname from dept where deptno in(select distinct deptno from
emp)"

DEPTNO DEPTNAME
------ ------------------------------------
A00    SPIFFY COMPUTER SERVICE DIV.
B01    PLANNING
C01    INFORMATION CENTER
D01    DEVELOPMENT CENTER
D11    MANUFACTURING SYSTEMS
D21    ADMINISTRATION SYSTEMS
E01    SUPPORT SERVICES
E11    OPERATIONS
E21    SOFTWARE SUPPORT
F22    BRANCH OFFICE F2
G22    BRANCH OFFICE G2
H22    BRANCH OFFICE H2
I22    BRANCH OFFICE I2
J22    BRANCH OFFICE J2
```

2.
```
select *from emp1 where emp_name like 'S%'"

EMP_NO EMP_NAME JOB MGR_ID  HIREDATE    SALARY       COMM         DEPT_NO
------- ----- ----- ------  --------   ------------ ------------ -----------
7369 SMITH    CLERK  7566   12/17/1990       8482.            0.          20
```

3.
```
select emp_name from emp1 where dept_no is not null

EMP_NAME
----------
SMITH
WARD
JONES
```

```
MARTIN
```

4.
```
select emp_name,salary from emp1 where salary between 10000 and 15000"

EMP_NAME   SALARY
---------- ------------
JONES            13721.
BLAKE            13583.
CLARK            10943.
JAMES            10393.
WILL             10447.
```

5.
```
select dept_no,avg(salary) as average_salary from emp1 group by dept_no having
avg(salary)>1000

DEPT_NO     AVERAGE_SALARY
----------- ---------------------------------
         10       10310.5000000000000000000000
         20        8913.0000000000000000000000
         30        8748.8000000000000000000000
         40       10447.0000000000000000000000
          -       15948.0000000000000000000000
```

6.
```
select distinct dept_no from emp1

DEPT_NO
-----------
         10
         20
         30
         40
```

7.
```
select emp_no,emp_name from emp1 where salary <6000 order by emp_name

EMP_NO      EMP_NAME
----------- ----------
       7902 FORD
       7844 TURNER
```

3.7 SET OPERATORS

OPERATOR	DESCRIPTION
UNION	Returns all rows retrieved by both the queries excluding common rows
UNION ALL	Returns all rows retrieved by both the queries including duplicate rows.
INTERSECT	Returns all common rows in both the queries.

EXCEPT	Returns all rows from the first query which are not existing in second query.

Example for UNION

```
select job from emp1 where dept_no=10 union select job from emp1 where dept_no=20
JOB
---------
ANALYST
CLERK
MANAGER
```

Example for UNION ALL

```
select job from emp1 where dept_no=10 union all select job from emp1 where
dept_no=20

JOB
---------
CLERK
MANAGER
CLERK
ANALYST
MANAGER
CLERK
```

Example For INTERSECT

```
select job from emp1 where dept_no=10 intersect select job from emp1 where
dept_no=20"

JOB
---------
CLERK
MANAGER
```

Example for EXCEPT

```
select job from emp1 where dept_no=10 except select job from emp1 where
dept_no=20"

JOB
---------

  0 record(s) selected.
```

3.8 BUILT-IN FUNCTIONS

A table in DB2 of SYSIBM’s schema against which we can test any functions.
The schema is SYSIBM.SYSDUMMY1

Example

```
Select 2*3 from SYSIBM.SYSDUMMY1;
```

3.8.1 CHARACTER FUNCTIONS:

FUNCTIONS
CHAR
ASCII
LOWER/LCASE
UPPER/UCASE
SUBSTR
REPLACE
TRANSLATE
LENGTH
LTRIM
RTRIM
LEFT
RIGHT
LOCATE
CONCAT

ASCII
ASCII returns the decimal representation in the database character set of the first character of char.

```
SELECT ASCII('A') FROM sysibm.sysdummy1;
--------------------------------------
                  65
```

CHR
CHR returns the character having the binary equivalent to n in either the database character set or the national character set.

```
SELECT CHR(65) FROM sysibm.sysdummy1;
---------------------------------------
                 -A
```

CONCAT
CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes CHAR, VARCHAR2, NCHAR,NVARCHAR2, CLOB, or NCLOB. The string returned is of VARCHAR2 datatype and is in the same character set as char1. This function is equivalent to the concatenation operator (||).

```
SELECT CONCAT('MIRACLE ','SOFTWARE') FROM sysibm.sysdummy1;
-----------------------------------------------
            MIRACLE SOFTWARE
```

LOCATE
This function search string for substring and returns an integer indicating the position of the

character in string that is the first character of this occurrence.

```
SELECT LOCATE('Miracle Software Systems','S',1,2)from sysibm.sysdummy1;
--------------------------------
            8
```

LENGTH
This function returns the length of the given string.

```
SELECT LENGTH('MIRACLE') FROM sysibm.sysdummy1;
LENGTH('MIRACLE')
------------------------------
7
```

LOWER
Convert the whole string into lower case.

```
SELECT LOWER('MIRACLE') FROM sysibm.sysdummy1;
LOWER('
------------------------
miracle
```

LTRIM
It trims the given string on LEFT side by removing the spaces on the left.

```
SELECT LTRIM('MIRACLE') FROM sysibm.sysdummy1;
LTRIM('T
------------------------------
MIRACLE
```

RTRIM
It trims the given string onRIGHT side by removing the spaces on the right.

```
SELECT RTRIM('MIRACLE') FROM sysibm.sysdummy1;
RTRIM('
---------------
MIRACLE
```

REPLACE
It returns a string with every occurrence of search string replaced with the replacement string.

```
SELECT REPLACE('THAT MIRACLE','THAT','TEAM') FROM sysibm.sysdummy1;
REPLACE('MI
------------------------
TEAM MIRACLE
```

SUBSTR
This function returns a portion of string beginning at the character portion and character length specified.

```
SELECT SUBSTR('MIRACLE SOFTWARE SYSTEMS',9,8) FROM sysibm.sysdummy1;
SUBSTR('
--------------------
SOFTWARE
```

TRANSLATE
TRANSLATE returns char with all occurrences of each character in 'from string' replaced by its corresponding character in 'to string'.

```
SELECT TRANSLATE('KIRACLE MOFTWARE','MS','KM') FROM sysibm.sysdummy1;
TRANSLATE('KIRAC
--------------------------------
MIRACLE SOFTWARE
```

UPPER
Converts the given string into uppercase.

```
SELECT UPPER('miracle') FROM sysibm.sysdummy1;
UPPER('
----------------------
MIRACLE
```

3.8.2 Numeric Functions

FUNCTIONS
ABS
CEIL
FLOOR
MOD
POWER
ROUND
SQRT
TRUNC

Examples

ABS
It returns the absolute value of the given integer.

```
select abs(52),abs(-52) from sysibm.sysdummy1"

1           2
----------- -----------
52          52
```

CEIL,FLOOR
It returns the smallest integer greater than or equal to the given integer.

```
SELECT CEIL(9.2),CEIL(-9.2),FLOOR(9.2),FLOOR(-9.2) FROM sysibm.sysdummy1

1    2    3    4
---- ---- ---- ----
10.  -9.   9. -10
```

MOD
It returns the remainder of the m divided by n, returns 0 if n is 0.

```
Syntax: SELECT MOD(m, n) FROM sysibm.sysdummy1;

SELECT MOD(14,5) FROM sysibm.sysdummy1;
4
```

POWER
Returns the first value raised the second value

```
Syntax: SELECT POWER(m, n) FROM sysibm.sysdummy1;

SELECT POWER(5,3) FROM sysibm.sysdummy1;
POWER(5,3)
----------------------
125
```

ROUND

```
select ROUND(1234.5666,2) FROM sysibm.sysdummy1

1
-----------
  1234.5700
```

SQRT
SQRT returns square root of n. The value cannot be negative. SQRT returns a "real" result.

```
SELECT SQRT(26) FROM sysibm.sysdummy1"

1
-------------------------
+5.09901951359278E+000
```

TRUNC

```
SELECT TRUNC(123.50,1),TRUNC(123.49,1) FROM sysibm.sysdummy1"

1       2
------- -------
123.50  123.40
```

3.8.3 DATE FUNCTIONS

FUNCTIONS
CURRENT_DATE
CURRENT_TIMESTAMP
DAY,MONTH, YEAR
HOUR, MINUTE, SECOND
MICROSECOND

DAYS
DAYNAME, MONTHNAME

CURRENT_DATE
Returns the current date in mm/dd/yyyy format.

```
SELECT CURRENT_DATE FROM sysibm.sysdummy1

1
----------
07/09/2010
```

CURRENT_TIMESTAMP
Returns current time and date in the session time zone.

```
SELECT CURRENT_TIMESTAMP FROM sysibm.sysdummy1"

1
--------------------------
2010-07-09-10.51.03.367290
```

DAY(date), MONTH(date), YEAR(date)
Returns the day, month, year of particular date.

```
SELECT DAY(CURRENT_DATE), MONTH(CURRENT_DATE),YEAR(CURRENT_DATE) FROM
sysibm.sysdummy1"

1           2           3
----------- ----------- -----------
9           7           2010
```

HOUR(time), MINUTE(time), SECOND(time)
Returns the hour, minute, second of particular time

```
SELECT hour(current_time), minute(current_time),second(current_time) FROM
sysibm.sysdummy1"

1           2           3
----------- ---------- -----------
10          55          33
```

MICRO SECOND(timestamp)
Returns the number of months between the two specified dates.

```
SELECT HOUR(current_time), MINUTE(current_time),SECOND(current_time) FROM
sysibm.sysdummy1"

1           2           3
---------- ----------- -----------
11           0           4
```

Examples

DESCRIPTION	DATE EXPRESSION
NOW	CURRENT DATE
TOMORROW/ NEXT DAY	CURRENT DATE + 1 DAY
SEVEN DAYS FROM NOW	CURRENT DATE + 7 DAYS
ONE HOUR FROM NOW	CURRENT TIME + 1 HOUR
THREE HOURS FROM NOW	CURRENT TIME + 3 HOURS
AN HALF HOUR FROM NOW	CURRENT TIME + 30 MINUTES
10 MINUTES FROM NOW	CURRENT TIME + 10 MINUTES
30 SECONDS FROM NOW	CURRENT TIME + 30 SECONDS

3.8.4 CONVERSION FUNCTIONS

FUNCTIONS
INT
CHAR
DATE
DECIMAL
NUMERIC

CHAR
To convert a date or timestamp value to a Varchar value.

```
"SELECT CHAR(current date) from sysibm.sysdummy1"

1
----------
07/09/2010
```

DATE
To convert a character date to a date value.

```
SELECT DATE('09-07-2010') FROM SYSIBM.SYSDUMMY1"

1
----------
09/07/2010
```

INT
To convert a character number into a number value.

```
SELECT INT(100.50) FROM sysibm.sysdummy1

1
-----------
100
```

DECIMAL

```
SELECT DECIMAL('100.50') FROM sysibm.sysdummy1"

1
-----------------
100.
```

3.8.5 GROUP FUNCTIONS

Group function work on a group of data to obtain aggregated values .

FUNCTIONS
AVG
COUNT
MIN
MAX
SUM

```
SELECT COUNT(SALARY), AVG(SALARY), MIN(SALARY), MAX(SALARY), SUM(SALARY) FROM
EMP1"

1           2                    3            4      5
--- --------------------------  --------  -------- ----------
13  9724.0000000000000000000000  4948.     15948.     126412.
```

3.9 Joins:

When the database is being normalized, data is distributed across multiple tables.
To retrieve the data from one or more tables using select statements is called as a Join.
The table given left to the comparison operator is called the left table and right to that is called the right table

Joins are classified as
CROSS JOIN
SELF JOIN
EQUI JOIN OR INNER JOIN
NON EQUI JOIN
OUTER JOIN

Self Join
Joining of a table to itself is called as a self join.
Example

```
SELECT e1.emp_name,e1.emp_name, e2.emp_name,e2.emp_name FROM emp1 e1, emp1 e2
WHERE e1.dept_no = e2.dept_no AND e1.emp_name LIKE '%S'
```

Equi Join or Inner Join

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

```
SELECT emp1.emp_name,department1.dept_no from emp1 inner join department1 on
emp1.dept_no=department1.dept_no order by emp1.emp_name
```

NON-EQUI JOIN
Using this type of join data can be retrieved from tables, which are not having any common columns.
Example

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

OUTER JOIN
Outer join is classified into three types.

LEFT OUTER JOIN
RIGHT OUTER JOIN
FULL OUTER JOIN

Left Outer Join
The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).

Syntax

```
SELECT column_name(s) FROM table_name1 LEFT JOIN table_name2 ON
table_name1.column_name=table_name2.column_name
```

Example

```
SELECT emp1.emp_name, department1.dept_no FROM emp1 LEFT JOIN department1 ON
emp1.dept_no=department1.dept_no order by emp1.emp_name
```

Right Outer Join
The RIGHT JOIN keyword Return all rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

Syntax

```
SELECT column_name(s) FROM table_name1 RIGHT JOIN table_name2 ON
table_name1.column_name=table_name2.column_name
```

Example

```
SELECT emp1.emp_name, department1.dept_no FROM emp1 RIGHT JOIN department1 ON
emp1.dept_no=department1.dept_no order by emp1.emp_name
```

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

Syntax

```
SELECT column_name(s) FROM table_name1 FULL JOIN table_name2 ON
table_name1.column_name=table_name2.column_name
```

Example

```
SELECT emp1.emp_name, department1.dept_no FROM emp1 FULL JOIN department1 ON
emp1.dept_no=department1.dept_no order by emp1.emp_name
```

3.10 SUB QUERIES and CO-RELATED SUB QUERIES

3.10.1 SUB QUERIES

A query embedded within another query is called as a subquery.
Using subqueries we can use the results of one query as the input for another query.
There are two types of sub queries
Single row sub query
Multiple row sub query

SINGLE ROW SUB QUERY
If the sub query returns only one single row then it is called as a single row sub query.
MULTIPLE ROW SUB QUERY
If the sub query returns more than one row then it is called as a multiple row sub query.

Example 1
Query to list the employee who is getting the maximum salary

```
SELECT ename FROM Employee WHERE salary = (SELECT max(salary) FROM Employee)
```

Example 2
Query to list the employees who are earning more than Blake

```
SELECTenameFROM Employee WHERE salary in (SELECT avg(salary) FROM Employee group
by job)
```

3.10.2 CO-RELATED SUB QUERIES

Example1
Write a query to list the employees who are earning more than their manager

```
SELECT ename FROM emp x WHERE sal > (SELECT sal FROM emp WHERE empno = x.mgr)
```

3.11 SEQUENCE

It is a auto generated number which will be unique.

Example 1

```
create sequence  sequence1
start with 1
increment by 1
no maxvalue
no cycle
cache 20
```

Example 2

```
create sequence  sequence2
start with 1
increment by 2
minvalue 10
maxvalue 100
no cycle
no cache
```

Using Sequence in a table

```
create table book(book_id int not null primary key,book_name
varchar(100),book_type varchar(100));
insert into book values(next value for sequence1,'Java','B.Tech');
```

3.12 VIEWS

A view is a logical or virtual table, which does not, exists physically in the database.
Views are also called as Dictionary Objects.
Advantages
Security – The confidential columns can be suppressed from viewing and manipulation.
Complexity can be avoided.
Logical columns can be created.
Application design becomes easier.
Views can be classified into two categories on the way they are created.

- Simple View
- Complex View

3.12.1 Simple View

If the view is created from a single table, it is called as a simple view. We can do DML Operation, if it is Simple View.

3.12.2 Complex View

If the view is created on multiple tables, it is called as a complex view.

Syntax:

```
CREATE VIEW <view name> AS <query>
```

Example 1

```
CREATE VIEW emp_view
 AS
 SELECT emp_name, salary FROM Emp1
```

Example 2

```
CREATE VIEW emp_view1
AS
```

```
SELECT ename, salary, salary*0.2,comm FROM Emp1
```

3.12.3 DROPPING A VIEW

Only the owner of the view can drop it.
Syntax :

```
DROP VIEW <view name>
```

Example

```
DROP VIEW emp_view
```

3.13 Index

An index can be created in a table to find data more quickly and efficiently. The CREATE INDEX statement is used to create indexes in tables. Indexes allow the database application to find data fast without reading the whole table. The users cannot see the indexes, they are just used to speed up searches/queries.

Syntax for INDEX creation:

```
CREATE INDEX index_name ON table_name (column_name)
```

Syntax for UNIQUE INDEX creation:

```
CREATE UNIQUE INDEX index_name ON table_name (column_name)
```

Example 1

```
create index dept_index on department1 (dept_name)
```

Example2

```
create unique index dept_index1 on department1 (dept_name)
```

Drop INDEX
Drop is used to delete index from Database.
Syntax:

```
DROP INDEX <INDEXNAME>
```

Example

```
drop index dept_index
```

4. Normalization

Normalization is a logical design method which minimizes data redundancy and reduces design flaws. It's a process of efficiently organizing data in a database.

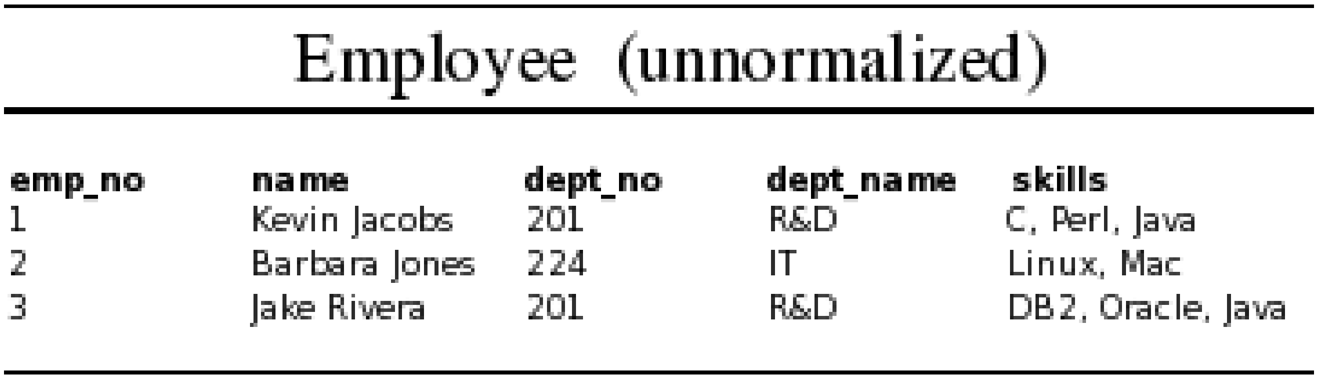
- Eliminating redundant data.
- Consists of applying various "normal" forms to the database design.
- The normal forms break down large tables into smaller subsets.

4.1 First Normal (1NF)

4.1.1 Each attribute must be atomic

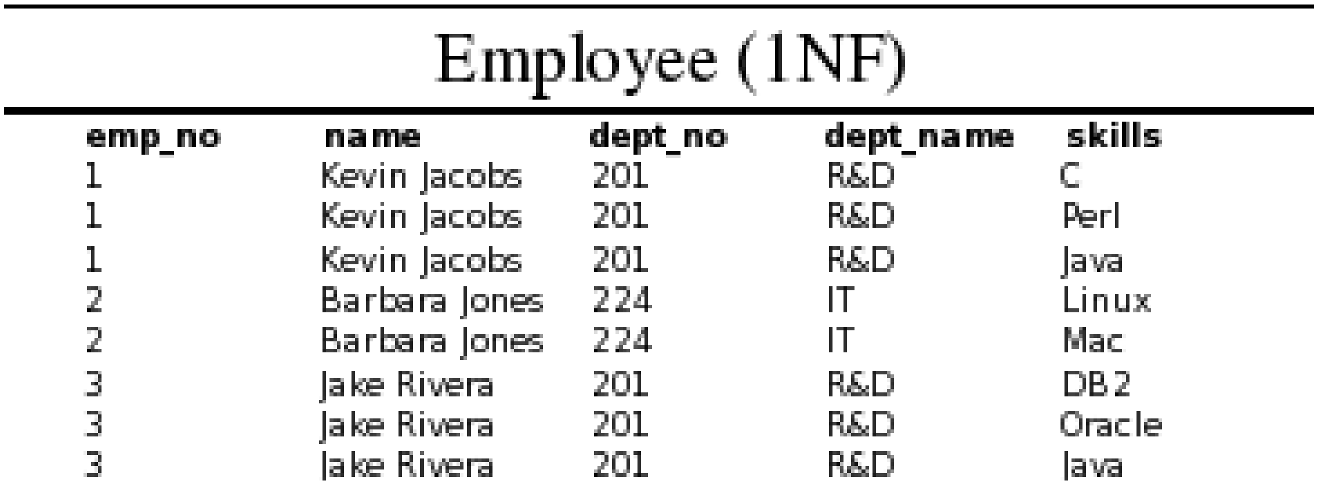
- No repeating columns within a row.
- No multi-valued columns.

1NF



Employee (unnormalized)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C, Perl, Java
2	Barbara Jones	224	IT	Linux, Mac
3	Jake Rivera	201	R&D	DB2, Oracle, Java

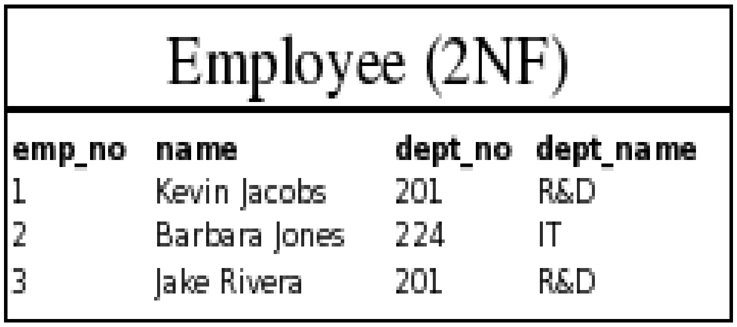


Employee (1NF)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

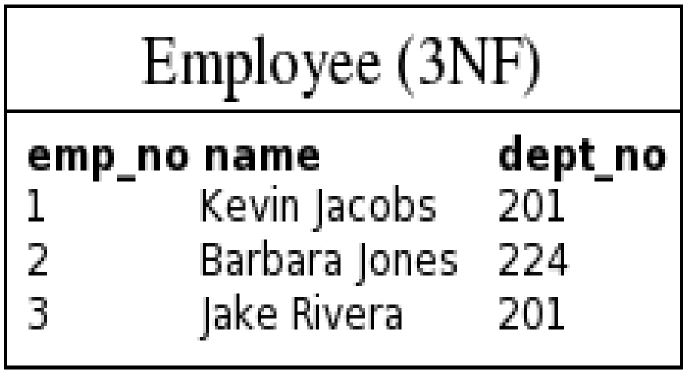
4.2 Second Normal Form (2NF)

Meet all requirements of the first normal form.
Remove subsets of data that apply to multiple rows of a table and place them in separate tables.
Create new relationships between these new tables and their predecessors through the use of



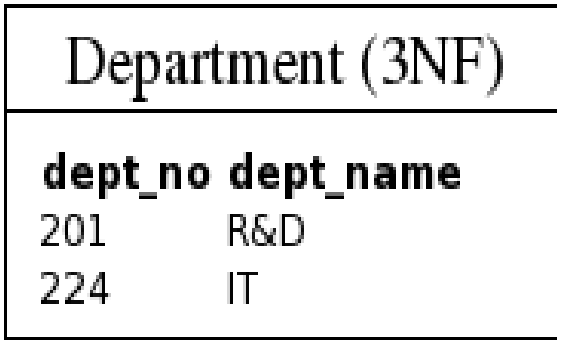
Employee (2NF)

emp_no	name	dept_no	dept_name
1	Kevin Jacobs	201	R&D
2	Barbara Jones	224	IT
3	Jake Rivera	201	R&D



Employee (3NF)

emp_no	name	dept_no
1	Kevin Jacobs	201
2	Barbara Jones	224
3	Jake Rivera	201



Department (3NF)

dept_no	dept_name
201	R&D
224	IT

foreign keys.

4.3 Third Normal Form (3NF)

Meet all requirements of the second normal form.
Remove columns that are not dependent upon the primary key.

4.4 Boyce-Codd Normal Form (BCNF)

Strengthens 3NF by requiring the keys in the functional dependencies to be superkeys (a column or columns that uniquely identify a row)

4.5 Fourth Normal Form (4NF)

Eliminate trivial multivalued dependencies.

4.6 Fifth Normal Form (5NF)

Eliminate dependencies not determined by keys.

5. PLSQL

SQL

In SQL there is no control on the flow of execution of the statements, what to do is can determined in SQL but when to do is not possible. By using SQL which we can interact with the database.

PLSQL

Procedural language / Structured Query Language. It is 3rd generation language there is a batch programming in PL/SQL. The basic unit of pl/sql is Block.
Block Types:
1. ANONYMOOUS BLOCK: These are meant for batch processing but they don't have any name.
2. NAMED BLOCK: These are meant for batch processing by using name objects are stored in database.

5.1 FUNCTIONS

It is a sub-program, which is storied in the database and whenever it is called, it does something and return some value. Functions can be used in SQL statement.
Syntax:

```
CREATE FUNCTION<func-name> (<param-name> <datatype>)
RETURNS <data type>/
BEGIN [ATOMIC / NON ATOMIC]
<SQL func. body>
RETURN <stmt. >
END
<stmt.> : 1. Var
          2. Compound SQL stmt. Like SELECT etc.
```

5.1.1 ALTER STATEMENT

ALTER FUNCTION

```
<function_name>
[FENCED / NOT FENCED]
[EXTERNAL ACTION / NO EXTERNAL ACTION]
[THREADSAFE / NOT THREADSAFE]
[NEW SAVEPOINT LEVEL]
```

5.1.2 Compiling

```
Db2 -td@ -vf  <path>
```

5.1.3 Executing

```
Select <func_name>(params) from sysibm.sysdummy1
Ex: select test (1,'a') from sysibm.sysdummy1
```

5.2 STORED PROCEDURE

It is Sub-Program, which is stored in the database and whenever it is called, it does something but does not return any value, but they could return indirectly through **OUT** parameter.
Syntax:

```
CREATE PROCEDURE <proc-name> (<param-name> <return-type> <datatype>)
BEGIN [ATOMIC / NON ATOMIC]
<SQL proc. body>
END
Begin: Indicates the beginning of the SP
Atomic: Indicates that if any unhandled exception occurs, all the stmts. will be
rolled back.
Non-Atomic: Indicates that if any unhandled exception occurs, the stmts. will not
be rolled back.
```

5.2.1 ALTER PROCEDURE

```
ALTER PROCEDURE <proc-name>
[FENCED / NOT FENCED]
[EXTERNAL ACTION / NO EXTERNAL ACTION]
[THREADSAFE / NOT THREADSAFE]
[NEW SAVEPOINT LEVEL]
```

5.2.2 Adv

Decrease Network traffic.

5.2.3 Compiling

```
db2 -td@ -vf <path>
```

5.2.4 Executing

```
CALL <SP_NAME> (Parameters)  -----  (Ex: Call Test(1,'a') )
```

5.3 CURSORS

A cursor is a memory context area, A PL/SQL program controls the context area using the cursor.

2-Types:

1. Implicit Cursor.
2. Explicit Cursor.

Implicit Cursor
SQl queries returns a single row, PL/SQL implicitly declares cursors for all DML statements.
Exception: NO_DATA_FOUND (or) TOO_MANY_ROWS.

Explicit Cursors: These are used in queries. That returns multiple rows explicit cursors are declared in [DECLARE SECTION] of PL/SQL program.

4-Stages of a cursor

```
DECLARE cursor;                     Ex: CURSOR <C_N> IS <select statement>;
OPEN cursor;                        Ex: OPEN <C_N>;
FETCHES the records into the cursor; Ex: FETCH <C_N> INTO <variables>;
CLOSING the cursor;                 Ex: CLOSE <C_N>;
```

Syntax:

```
DECLARE <cur_name> CURSOR
[WITH HOLD / WITH RETURN]
[TO CALLER / TO CLIENT] FOR <select stmt. >
OPEN <cur_name>: Opens the cursor.
FETCH [FROM] <cur_name> INTO <variables>: Fetches the values of the result set
into the corresponding variables.
CLOSE <cur_name> [WITH RELEASE]: Closes the cursor.
```

Explicit Cursors Attributes

```
1.  %NOTFOUND  ---------> True - the last fetch failed.
2.  %FOUND ----------------> True - the last fetch succeeded.
3.  %ROWCOUNT---------> Returns the number of rows fetched by the cursor so far.
4.  %ISOPEN----------> True - If cursor is open. (or) False - If cursor is closed.
```

5.4 TRIGGERS

A trigger is a block of statements which are fired automatically when DML operation take place. Triggers are always associated with database tables.

Trigger types

1. Row-Level Trigger.
2. Statement Level Trigger.
3. Before Trigger
4. After Trigger.

Row-Level Trigger

These triggers will be fired for each row with proportional to the number of rows. Those are effected the DML Statements.

Statement Level Trigger

These triggers are fired once for each row statement regardless with the number of rows those are effected by the DML statements.

Before Triggers

These triggers are fired before the execution of the DML statements.

After Triggers

These triggers are fired after the execution of the DML statements.

SYN: create trigger <trigger-name> before/after insert or delete or update on <table name> for each row/ statement.

Syntax & Ex

```
CREATE TRIGGER Totalsal_insert
AFTER INSERT ON Emp REFERENCING new AS new_sal
FOR EACH ROW
```

5.4.1 TRIGGER FOR INSERTION

```
BEGIN
UPDATE Dept1 SET totalsal = totalsal + new_sal.sal WHERE Deptno = new_sal.Deptno;
END
@
```

5.4.2 TRIGGER FOR UPDATION

```
CREATE TRIGGER Totalsal_update
AFTER UPDATE ON Emp REFERENCING NEW AS new_sal OLD AS old_sal
FOR EACH ROW
BEGIN
UPDATE Dept1 SET totalsal = totalsal - old_sal.sal + new_sal.sal
WHERE Deptno = new_sal.Deptno
END
@
```

5.4.3 TRIGGER FOR DELETION

```
CREATE TRIGGER Totalsal_delete
AFTER DELETE ON Emp REFERENCING OLD AS old_sal
FOR EACH ROW
BEGIN
UPDATE Dept1 SET totalsal = totalsal - old_sal.sal WHERE Deptno = old_sal.Deptno
END
@
```

5.5 FOR statement

The FOR loop will be used very often as a simple read-only cursor. You've just to specify the select statement after the FOR <loop name> AS keywords. Optional there's the possiblity to specify a name for this cursor. If you don't use the cursor option DB2 will generate a unique name for this cursor. The end of the looped section is marked with the END FOR statement.

It's not possible to reference the defined cursor outside of this for loop. Accessing this cursor by the OPEN, FETCH, or CLOSE statement is not supported.

Syntax:

```
for <loop-name> as [<cursor-name>] cursor [with hold] for <select-statement>
 do
   <inside-loop-logic>
 end for;
```

Example:

```
create table mytab (zipcode char(4), city char(30))!
create table mytab2 (zip_and_city char(35))!

insert into mytab values ('9500','Paris')!

create procedure myproc
begin
  for i as mycursor cursor for select zipcode, city from mytab
    do
      insert into mytab2 values (zipcode concat ' ' concat city);
 end for;
end!
```

5.6 IF statement

The IF statement can be used instead of the SEARCHED-CASE statement. With the IF statement it's possible to enter into some logic based on the status of the IF condition.
You've to specifiy one IF-THEN condition and one ELSE clause. Optionally there are one ore multiple ELSEIF clauses possible. Note that ELSEIF is one word and don't forget the END IF clause to to specifiy the end of the IF statement.

Example 1:

```
create procedure myproc (in v_var01 integer, out v_var02 varchar(50))
 begin
    if v_var01 < 0 then
      set v_var02='Variable is lower then 0';

    elseif v_var01 >= 0 and v_var01 < 10 then
      set v_var02='Variable is between 0 and 9';

    elseif v_var01 >= 10 and v_var01 < 20 then
      set v_var02='Variable is between 10 and 19';

    else
      set v_var02='Variable is 20 or higher';

    end if;
 end!
```

Example 2:
This example shows an User Defined Function written in Inline SQL PL:

```
create function convert_to_eur(value decimal(6,2), currency varchar(3))
returns decimal(6,2)
language sql
deterministic
contains sql
begin atomic

  declare output decimal(6,2);

  if currency = 'DEM' then
     set output = value/1.95583;
  elseif currency = 'BEF' then
     set output = value/40.3399;
  elseif currency = 'FIM' then
     set output = value/5.94573;
  elseif currency = 'FRF' then
     set output = value/6.55957;
  elseif currency = 'GRD' then
     set output = value/340.750;
  elseif currency = 'IEP' then
     set output = value/0.787564;
  elseif currency = 'ITL' then
     set output = value/1936.27;
  elseif currency = 'LUF' then
     set output = value/40.3399;
  elseif currency = 'MTL' then
     set output = value/0.429300;
  elseif currency = 'NLG' then
     set output = value/2.20371;
  elseif currency = 'ATS' then
     set output = value/13.7603;
  elseif currency = 'PTE' then
     set output = value/200.482;
  elseif currency = 'SIT' then
     set output = value/239.640;
  elseif currency = 'ESP' then
     set output = value/166.386;
  elseif currency = 'CYP' then
     set output = value/0.585274;
  end if;

return output;
end!
```