Unit 4. DATA BASE MANAGEMENT SYSTEM (DB2)

- **BASIC OBJECTS AND OPERATORS**
- **DATA DEFINITION**
- **DATA MANIPULATION**
    - **DATA MANIPULATION RETRIEVAL OPERATIONS**
    - **DATA MANIPULATION UPDATE OPERATIONS**
- **CATALOG**
- **VIEWS**
- **SECURITY AND AUTHORIZATION**
- **INTEGRITY**

Figure: 4.1 Database management system (DB2)

Notes:

BASIC OBJECTS & OPERATORS SUPPORTED BY DB2

DATA TYPES

- **Numeric**
- **String**
- **Date /Time**
- **Graphic**

LITERALS

- **Integer**
- **Decimal**
- **Date/Time/Timestamp**

SCALAR OPERATORS AND FUNCTIONS

- **Numeric operators**
- **Concatenation**
- **Char**
- **Date**
- **Days**
- **Decimal**
- **Sub string**
- **Timestamp**
- **Integer**

Figure: 4.2 Basic Objects and Operators Supported by DB2

DATA TYPES

Numeric data:

INTEGER	Four-byte binary integer, 31 bit and sign
SMALLINT	Two byte binary integer, 15 bit and sign
DECIMAL (p, q)	Packed decimal number, p digits and sign (0<p<32), with assumed decimal point, q digits from the right (0<=q<=p), occupying (p+1)/2 or (p+2)/2 bytes where p can be odd or even
NUMERIC	Max. 18 digits.

String Data type:

CHARACTER (n)	fixed length string of exactly n 8 bit characters (0<n<255), occupying n bytes
VARCHAR (n)	varying length string up to 8 bit characters (0<n), occupying n+2 bytes (2 bytes for a hidden length field)

Figure: 4.3 Data Types

DATA TYPES (Cont…)

Date/Time Data Type

DATE	Date, represented as a sequence of 8 unsigned, packed decimal digits (yyyymmdd), occupying 4 bytes.
TIME	Time, represented as a sequence of 6 unsigned packed decimal digits (hhmmss), occupying 3 bytes.
TIMESTAMP	Timestamp, (combination of date and time accurate to the nearest microsecond) represented as a sequence of 20 unsigned packed decimal digits (yyyymmddhhmmssnnnnnn), occupying 10 bytes

Graphic Data Type

GRAPHIC (n)	Fixed length string of exactly ‘n’ 16 bit characters (0<n<128), occupying 2n bytes.
VARGRAPHIC (n)	Varying length string of up to ‘n’ 16 bit characters (0<n), occupying (2n+2) bytes (2 bytes for hidden length field

Figure: 4.4 Data Types (Cont…)

LITERALS

INTEGER	Written as a signed or unsigned decimal integer, with no decimal point Ex. 4 -95 +365 0
DECIMAL	Written as a signed or unsigned decimal integer, with decimal point Ex. 4. -95.7 +365.05 0.007
DATE	Written as a character string literal of the form mm/dd/yyyy, enclosed in single quotes. Ex. '1/18/1941'
TIME	Written as a character string literal of the form hh:mm AM or PM, enclosed in a single quotes. Ex. '10:00 AM' '9:30 PM'
TIMESTAMP	Written as a character string literal of the form yyyy-mm-dd-hh.mm.ss.nnnnnn enclosed in single quotes. Ex. '1990-4-28-12.00.00.000000' '1944-10-17-18.30.45'

Figure: 4.5 Literals

DATA DEFINITION

- **BASE TABLES**
    - **CREATE TABLE**
    - **ALTER TABLE**
    - **DROP TABLE**
- **INDEXES**

Figure: 4.6 Data Definition

BASE TABLE

A base table is an important special case of the more general concept “table”. It is defined as “A table in a relational system consists of a row of column headings, together with zero or more rows of data values (different numbers of data rows at different times)

For a given table:

The column-heading row specifies one or more columns (giving, among other things, a data type for each).

Each data row contains exactly one scalar value for each of the columns specified in the column-heading row. Furthermore, all the values in a given column are of the same data type, namely the data type specified in the column-heading row for that column.

Figure: 4.7 Base Tables

CREATE TABLE

CREATE TABLE <tablename>(
Column1 Datatype(Size) Null charasteristics , …..)

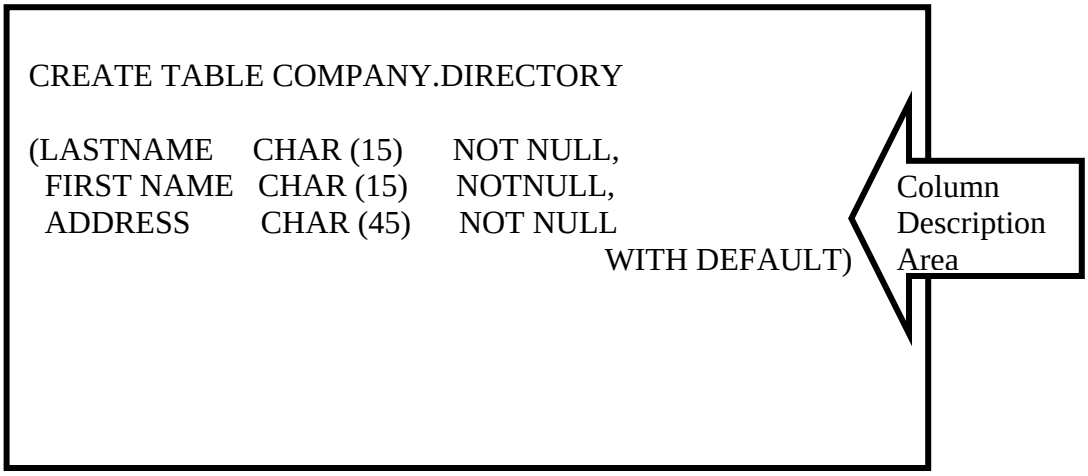

Figure: 4.8 Create table

CREATE TABLE (Cont…)

This CREATE Statement tells DB2 that:

- **The owner of the table is to be COMPANY.**
- **The simple name of the table is to be DIRECTORY.**
- **The name of the first column will be LASTNAME.**
- **Only character data is to be allowed in the first column.**
- **All values in the first column will be stored with 15 characters.**
- **The first column must always have a value.**
- **The name of the second column will be FIRSTNAME.**
- **Only character data is to be allowed in the second column.**
- **If a first name is not available for a row DB2 is to mark it null (unknown).**
- **The name of the third column will be ADDRESS.**
- **Only character data is to be allowed in the third column.**
- **If an ADDRESS is not available for a given row, DB2 is to fill the ADDRESS column with spaces.**

Figure: 4.9 Create table (Cont…)

CREATE TABLE (Cont…)

A Table can have any number of unique keys, but only one defined primary key

Types

Simple –1 Column
Contains Unique Values

Compound – Multiple Columns
A table may have more than one unique key

Primary key – One per table
All columns must be NOT NULL

Foreign key –
A column or set of columns that contains values from some table's unique key.

Figure: 4.10 Create table (Cont…)

CREATE TABLE (Cont…)

CREATE TABLE Statement

```
CREATE  TABLE  S
( S#          CHAR(5)     NOT NULL,
  SNAME     CHAR (20)   NOT NULL,
  STATUS    SMALLINT  NOT NULL,
  CITY        CHAR(15)   NOT NULL,
  PRIMARY KEY ( S# ));

CREATE  TABLE  P
( P#          CHAR(6)      NOT NULL,
  PNAME     CHAR(20)    NOT NULL,
  COLOR     CHAR(6)      NOT NULL,
  WEIGHT   SMALLINT  NOT NULL,
   CITY       CHAR(15)   NOT NULL,
PRIMARY KEY ( P# ));
```

EXAMPLE FOR CREATING A TABLE USING PRIMARY KEY AND A FOREIGN KEY:

```
CREATE  TABLE  SP
( S#          CHAR (5)       NOT NULL,
  P#          CAHR (6)       NOT NULL,
  QTY       NUMERIC(6,2) NOT NULL,
  FOREIGN KEY ( S# ) REFERENCES S,
  FOREIGN KEY ( P# ) REFERENCES P);
```

By defining S#, P# as a foreign key, we are telling to DB2 that the row for a given S, P is related to the row in the SP table.

Figure: 4.11 Create table (Cont…)

CREATE TABLE (Cont…)

PRIMARY KEY, FOREIGN KEY

SP

FK

S#	P#	QTY

S

PK

S#	SNAME	STATUS	CITY

Figure: 4.12 Create table (Cont…)

CREATE INDEXES

Like base tables, INDEXES are created and dropped using SQL data definition statements. However, CREATE INDEX and DROP INDEX (also ALTER INDEX are the only statements in the SQL language that refer to indexes at all)

CREATE INDEX takes the general form:

CREATE (UNIQUE) INDEX indexname
ON base-table (column (order) (, column (order))….)
(Other parameters)

The optional 'other parameters' have to do with physical storage matters, as in CREATE TABLE. The order specifies its either Ascending or Descending. If neither both are specified it assumes Ascending as default.

The left-to-right sequence of naming Columns in the CREATE INDEX statement corresponds to major- to – minor ordering in the usual way.

Figure: 4.13 Create Indexes

INDEXES (Cont….)

For e.g.

CREATE INDEX INDX ON TAB (P, Q DESC, R)

The statement creates an index called INDX on base table TAB in which entries are ordered by ascending R-value within descending Q value within ascending P value. The columns P, Q and R need not be contiguous, nor need they all are of the same data type.

Index once created is automatically maintained by the Data Manager to reflect updates on the tables, until the indexes are dropped.

The UNIQUE option in CREATE INDEX specifies that no two rows in the index base table will be allowed to take on the same value for the indexed column or column combination at the same time. Indexes, like base tables can be created and dropped at any time. However an attempt is made to create a UNIQUE INDEX on a non-empty table that already violates the uniqueness constraint will fail.

Figure: 4.14 Indexes (Cont…)

ALTER TABLE

As you create a new base table at any time, via CREATE TABLE, so an existing base table can be altered at any time by the addition of a new column at the right, via ALTER TABLE.

ALTER TABLE base-table
ADD column data type (NOT NULL WITH DEFAULT)
For e.g.

ALTER TABLE SP ADD PRICE SMALLINT

This statement adds a PRICE column to the SP table. All existing SP rows are extended from four columns to five, the value of the new fifth column is null in every case (it would have been zero if NOT NULL WITH DEFAULT had been specified). Note that the unqualified specification NOT NULL – i.e. with further specification WITH DEFAULT omitted is allowed in ALTER TABLE. The expansion of the existing rows just described is not physically performed at the time the ALTER TABLE is executed; all that happens at that time is that the description of those rows in the catalog changes.

Figure: 4.15 Alter Table

ALTER TABLE (Cont…)

Therefore for a given row in the altered table

1. The next time it is read from the disk, DB2 appends the additional NULL or NOT NULL default value before passing it to the user.
2. The next time it is written to the disk, DB2 writes the physically expanded version (unless the additional value is still null or nonnull default value, in which case the expansion still does not occur).

ALTER TABLE also allows Primary Key and Foreign (but not alternate) key specifications to be added to or removed from a given base table.

Figure: 4.16 Alter Table (Cont…)

DROP TABLE / INDEX

DROP TABLE

An existing base table can be deleted at any time by means of the DROP TABLE statement.

DROP TABLE base-table

The specified base table is removed from the system (more precisely, the description of that table is removed from the catalog). All indexes and views defined on that base table are automatically dropped (all foreign key specifications that refer to that base table are also automatically dropped).

DROP INDEX

DROP INDEX index-name.

Figure: 4.17 Drop Table

DATA MANIPULATION I: UPDATE OPERATIONS

- **INSERT**
  - **SINGLE ROW INSERT**
  - **SINGLE ROW INSERT WITH COLUMN NAME OMITTED**
- **UPDATE**
  - **SINGLE ROW UPDATE**
  - **MULTIPLE ROW UPDATE**
  - **UPDATE WITH A SUBQUERY**
- **DELETE**
  - **SINGLE ROW DELETE**
  - **MULTIPLE ROW DELETE**
  - **DELETE WITH A SUBQUERY**

Figure 4.18 Data Manipulation I: Update Operations

INSERT

The INSERT has a general format

INSERT
INTO tablename ((Column1) (, Column2….))
VALUES (LiteraI) (, Litera)….));

In this given format, a row is INSERTed into the table having the specified values for the specified columns, the I th literal in the list of literals corresponds to the Ith column in the list of columns.

Single Row INSERT

The query
‘Add part P7 (city Athenes, weight 24, name and color at present unknown) to table P.

INSERT
INTO P (P#, CITY, WEIGHT)
VALUES (‘P7’, ‘Athenes’, 24);

The values for PNAME and COLOR depends on their definition during CREATE or ALTER table statement:

NOT NULL WITH DEFAULT: The column is set to be appropriate not null default values. DB2 automatically places one of the following not null default values in that position.

Figure: 4.19 Insert

INSERT (Cont….)

- **Zero for numeric columns**
- **Blanks for fixed length string columns**
- **Empty (zero-length string) for varying length string columns**
- **CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP is used in appropriate date, time and timestamp columns.**

NOT NULL

The INSERT fails (database remains unchanged).

Note that the left-to-right order in which columns are named in the INSERT statement does not have to be the same as the left-to-right order in which they appear in the table.

Single Row INSERT with column name omitted:

The query
'Add part P7 ('Athenes', 24) to table P'.

```
INSERT
INTO P
VALUES ('Athenes', 24);
```

Figure: 4.20 Insert (Cont…)

UPDATE

The UPDATE statement has general form:

UPDATE Tablename
SET (column = scalar-expression
(, Column = scalar –expression)…
(WHERE condition);

All rows in a table that satisfy the condition are UPDATEd in accordance with the assignments (column = scalar-expression)
In the SET clause.

Single Row UPDATE:

The query
‘ Change the COLOR of the part P2 to yellow, increase its WEIGHT to 5 and SET its CITY to unknown (i.e. NULL)’.

UPDATE P
SET COLOR = ‘YELLOW’
WEIGHT = WEIGHT + 5,
CITY = NULL
WHERE P# = ‘P2’;

Figure: 4.21 Update

UPDATE (Cont…)

Multiple Row UPDATE:

The query
'Double the STATUS of all the suppliers in London'.

```
UPDATE  S
SET STATUS = 2 * STATUS
WHERE CITY = 'London'
```

UPDATE with a sub query:

The query
'Set the shipmen quantity to 0 for all suppliers in London'.

```
UPDATE  SP
SET QTY = 0
WHERE 'London' =
                (SELECT CITY
                  FROM S
                   WHERE S.S# = SP.S#)
```

Figure: 4.22 Update (Cont…)

DELETE

The DELETE statement has the general form

```
DELETE
FROM tablename
[WHERE condition];
```

All rows in ‘table’ that satisfies ‘condition’ (or all rows, if the WHERE clause is omitted) are DELETEd.

Single-Row DELETE.

The query ‘ Delete supplier S5’

```
DELETE  FROM S
 WHERE S# = ‘S5’;
```

Multiple-Row DELETES.
The query
’Delete all shipments with quantity greater than 300’.

```
DELETE     FROM  SP
 WHERE QTY > 300;
```

Figure: 4.23 Delete

DELETE (Cont…)

Multiple-Row DELETES.
The query ’Delete all shipments’.

```
DELETE  FROM  SP;
```

DELETE with a Sub query. Delete all shipments for suppliers in London.

```
DELETE
      FROM SP
         WHERE ‘London’ = (SELECT CITY FROM S WHERE
                                   S.S# = SP.S#)
```

Figure 4.24 Delete (Cont…)

DATA MANIPULATION II: RETRIEVAL OPERATIONS

- **INTRODUCTION**
- **SIMPLE QUERIES**
  - **SIMPLE RETRIEVAL**
  - **RETRIEVAL OF COMPUTED VALUES**
  - **QUALIFIED RETRIEVAL**
  - **RETRIEVAL USING BETWEEN**
  - **RETRIEVAL USING IN**
  - **RETRIEVAL USING LIKE**
- **JOIN QUERIES**

Figure: 4.25 Data Manipulation I: Retrieval Operations

DUMMY TABLES

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	10
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Figure: 4.26 Dummy Tables

SIMPLE QUERIES

SQL provides four Data Manipulation statements. SELECT, INSERT,UPDATE and DELETE.

Simple queries

Ex. The queries ‘get supplier number and status for the suppliers in Paris’, which can be expressed in SQL as follows

SELECT S#, STATUS
FROM S
WHERE CITY = ‘PARIS

Result

S#	STATUS
S1 S2	10 30

The example illustrates the SQL SELECT statement –“SELECT specified values FROM a specified table WHERE some specified condition is true”.

Figure: 4.27 Simple Queries

SIMPLE QUERIES(Cont…)

SIMPLE RETRIEVAL

Ex. The query "gets part numbers for all parts supplied."

SELECT P# FROM SP;

RESULT:

P#
P1
P2
P3
P4
P5
P6
P1
P2
P2
P2
P4
P5

Notice the duplication of part numbers in this result. DB2 does not eliminate duplicate rows from the result of a SELECT statement unless the user explicitly requests it to do so via keyword DISTINCT.

Figure: 4.28 Simple Queries (Cont…).

SIMPLE QUERIES (Cont…)

RETRIEVAL WITH DUPLICATE ELIMINATION:

The query "Get part numbers for all parts supplied, with redundant duplicates eliminated."

SELECT DISTINCT P# FROM SP;

RESULT:

P#
P1
P2
P3
P4
P5
P6

In this example each row contains a single scalar value, the effect of DISTINCT specification is therefore to eliminate duplicate scalar values. DISTINCT means eliminating duplicate rows.

Figure: 4.29 simple queries (Cont…)

SIMPLE QUERIES (Cont….)

RETRIEVAL OF COMPUTED VALUES

This example gives the use of DISTINCT with rows containing more than one scalar value. The query ‘get the part no and the weight of the part in grams’.

```
SELECT  P#,  ‘ Weight in grams = ‘, WEIGHT * 454
FROM P
```

Result

P#	Weight in grams =	
P1	Weight in grams =	5448
P2	Weight in grams =	4546
P3	Weight in grams =	4567
P4	Weight in grams =	3467
P5	Weight in grams =	5675
P6	Weight in grams =	5645

SIMPLE RETRIEVAL USING (“ SELECT * ”)

Ex. The query to get full details of all the suppliers:

```
SELECT * FROM S
```

Result A copy of entire S table has been displayed.

The * or asterisk is shortened for a list of all column names in the tables named in the FROM clause, in the left to right order in which those columns appear in the relevant tables.

Figure: 4.30 Simple Queries (Cont…)

SIMPLE QUERIES (Cont…)

QUALIFIED RETRIEVAL

The query "get supplier numbers for suppliers in Paris with STATUS greater than 20"

```
SELECT S#
FROM S
WHERE CITY = 'PARIS'
AND STATUS > 20
```

Result

S#
S3

Qualified retrieval is been done based on WHERE clause.

Figure: 4.31 Simple Queries (Cont…)

SIMPLE QUERIES (Cont…)

RETRIEVAL WITH ORDERED

The query “gets suppliers number and status for suppliers in Paris, in descending order of status “

```
SELECT  S#, STATUS
FROM S
WHERE CITY= ‘PARIS’
ORDERED BY STATUS DESC
```

Result

S#	STATUS
S3	30
S2	10

Figure: 4.32 Simple Queries (Cont.)

SIMPLE QUERIES (Cont….)

RETRIEVAL USING BETWEEN

The query " gets parts whose Weight is in the range 16 to 19 inclusive".

```
SELECT P#, PNAME, COLOR, WEIGHT, CITY
FROM P
WHERE WEIGHT BETWEEN 16 AND 19
```

Result

P#	PNAME	COLOR	WEIGHT	CITY
P2	BOLT	GREEN	17	PARIS
P3	SCREW	BLUE	17	ROME
P6	COG	RED	19	LONDON

The BETWEEN condition is really short hand condition involving two individual comparisons "ANDed" together. The foregoing SELECT statement is equivalent to the following

```
SELECT P#, PNAME, COLOR, WEIGHT, CITY
FROM P
WHERE WEIGHT >= 16
AND WEIGHT <= 19;
```

Figure: 4.33 Simple Queries (Cont.)

SIMPLE QUERIES (Cont…)

RETRIEVAL USING NOT BETWEEN

For ex.

```
SELECT P#, PNAME, COLOR, WEIGHT, CITY
FROM P
WHERE WEIGHT NOT BETWEEN 16 AND 19
```

Result

P#	PNAME	COLOR	WEIGHT	CITY
P1	NUT	RED	12	LONDON
P4	SCREW	RED	14	LONDON
P5	CAM	BLUE	12	PARIS

Like the BETWEEN condition, the NOT BETWEEN condition can be regarded merely as shorthand for another condition that does not use NOT BETWEEN.

Figure: 4.34 Simple Queries (Cont.)

SIMPLE QUERIES (Cont…)

RETRIEVAL USING IN

The query gets parts whose Weight is any one of the following
12, 16, 17

```
SELECT P#, PNAME, COLOR, WEIGHT, CITY
FROM P
WHERE WEIGHT IN (12,16,17)
```

Result

P#	PNAME	COLOR	WEIGHT	CITY
P1	NUT	RED	12	LONDON
P2	BOLT	GREEN	17	PARIS
P3	SCREW	BLUE	17	ROME
P5	CAM	BLUE	12	PARIS

IN, like BETWEEN is really like shorthand. An IN condition is logically equivalent to a condition involving a sequence of individual comparisons all “OR” together.

For e.g. the foregoing SELECT statement is equivalent to the following

```
SELECT P#, PNAME, COLOR, WEIGHT, CITY
FROM P
WHERE WEIGHT = 12
OR WEIGHT = 16
OR WEIGHT = 17;
```

Figure: 4.35 Simple Queries (Cont.)

SIMPLE QUERIES (Cont…)

RETRIEVAL USING NOT IN

Ex.

```
SELECT P#, PNAME, COLOR, WEIGHT, CITY
FROM P
WHERE WEIGHT NOT IN (12,16,17)
```

Result

P#	PNAME	COLOR	WEIGHT	CITY
P4	SCREW	RED	14	LONDON
P6	COG	RED	19	LONDON

RETRIEVAL USING LIKE

Get all parts whose Names begin with the letter C

```
SELECT P#, PNAME, COLOR, WEIGHT, CITY
FROM P
WHERE PNAME LIKE 'C%'
```

Result

P#	PNAME	COLOR	WEIGHT	CITY
P5	CAM	BLUE	12	PARIS
P6	COG	RED	19	LONDON

Figure: 4.36 Simple Queries (Cont.)

JOIN QUERY

The ability to “JOIN” two or more tables is one of the most powerful feature of relational systems. A JOIN can be defined as “a query in which data is retrieved from more than one table”.

Types of JOIN

- **INNER JOIN**
- **OUTER JOIN**
    - **RIGHT OUTER JOIN**
    - **LEFT OUTER JOIN**

Figure: 4.37 Join Query

TABLES

TABLE EMP

EMPNO	LASTNAME
000010	JOHN
000150	ROBERT
000020	TOM
000250	SMITH
000100	BETTY
000070	JIL
200140	JACK

TABLE DEPT

DETPNAME	MGRNO
SPCOMP	000010
DEVP	-
SW	000100
ADMIN	000070
BRANOFF	-

Figure: 4.38 Tables

JOIN QUERY

We can find the name of a department manager by looking in the EMPNO column for values that are contained in the MGRNO column. When we find a matching value, reading across the row to the name column provides us with the manager's name.

The MGRNO and EMPNO columns relate these two tables. There are departments with unknown managers, and there are employees that are not managers.

SIMPLE JOIN

```
SELECT LASTNAME, DEPTNAME
FROM EMP, DEPT
WHERE EMPNO = MGRNO
```

LASTNAME	DEPTNAME
JOHN	SPCOMP
BETTY	SW
JIL	ADMIN

Figure: 4.39 Join Query

JOIN QUERY (Cont…)

INNER JOIN

```
SELECT LASTNAME, DEPTNAME
FROM EMP INNER JOIN DEPT
ON EMPNO = MGRNO
```

LASTNAME	DEPTNAME
JOHN	SPCOMP
BETTY	SW
JIL	ADMIN

When JOIN key word is used in the FROM clause the join predicates must be written in an ON clause, which is an extension to the FROM clause.

Figure: 4.40 Join Query (Cont.)

JOIN QUERY (Cont…)

OUTER JOIN

It allows unmatching rows from either or both tables to be included in the result of join.

EMP TABLE

EMPNO	LASTNAME
000010	JOHN
000150	ROBERT
000020	TOM
000250	SMITH
000100	BETTY
000070	JIL
200140	JACK

DEPT TABLE

DETPNAME	MGRNO
SPCOMP	000010
DEVP	-
SW	000100
ADMIN	000070
BRANOFF	-

Figure: 4.41 Join Query (Cont….)

JOIN QUERY (Cont…)

RIGHT OUTER JOIN

Each unpaired row of the right table is concatenated with the null (non-existent) row from the left table.

```
SELECT LASTNAME, DEPTNAME
FROM EMP RIGHT OUTER JOIN DEPT
ON EMPNO = MGRNO
```

LASTNAME	DEPTNAME
JOHN	SPCOMP
BETTY	SW
JIL	ADMIN
-	DEVP
-	BRANOFF

The join specification, in the case, RIGHT OUTER JOIN, is in the middle of two table names. One of the names is on the right side of the join specification and one is on the left. When using the join specification RIGHT OUTER JOIN, we are telling DB2 to return us in addition to the related rows between our tables, all unmatching rows from the table on the right of the join specification.

Figure: 4.42 Join Query (Cont…)

JOIN QUERY (Cont…)

LEFT OUTER JOIN

Each unpaired row of the left table is concatenated with the null (non-existent) row from the right table.

```
SELECT LASTNAME, DEPTNAME
FROM EMP LEFT OUTER JOIN DEPT
ON EMPNO = MGRNO
```

LASTNAME	DEPTNAME
JOHN	SPCOMP
BETTY	SW
JIL	ADMIN
ROBERT	-
TOM	-
SMITH	-
JACK	-

Figure: 4.43 Join Query (Cont…)

DATA MANIPULATION III: RETRIEVAL OPERATIONS

- **SUBQUERIES**
- **SUBQUERY WITH MULTIPLE LEVEL OF NESTING**
- **CORRELATED SUBQUERY**
- **AGGREGATE FUNCTIONS**
    - **AGGREGATE FUNCTION IN A SELECT CLAUSE**
    - **AGGREGATE FUNCTION IN A SUBQUERY**
- **USE OF GROUP BY**
- **USE OF HAVING**
- **UNION**
- **QUERY INVOLVING UNION**

Figure: 4.44 Data Manipulation III: Retrieval Operations

SUBQUERY

A sub query is a SELECT FROM WHERE expression that is nested inside another such expression. Sub queries are typically used to represent the set of values to be searched by means of IN condition. Ex. A query ‘gets supplier names for suppliers who supply part P2’.

```
SELECT SNAME
FROM S
WHERE S# IN
            (SELECT S#
              FROM SP
              WHERE P# = ‘P2’)
```

Result

SNAME
SMITH
JONES
BLAKE
CLARK

Figure: 4.45 Sub query

SUBQUERY WITH MULTIPLE LEVELS OF NESTING

Ex. The query ‘ get suppliers name for suppliers who supply at least one RED part’.

```
SELECT SNAME
FROM S
WHERE S# IN
            (SELECT S#
             FROM SP
             WHERE P# IN
                        (SELECT P#
                         FROM P
                         WHERE COLOR = ‘RED’))
```

Result

SNAME
SMITH
JONES
BLAKE
CLARK

The innermost sub query evaluates to the set (P1 P4 P6), the next outermost sub query evaluates in turn to a set (S1 S2 S4). The last outermost sub query evaluates to the final result shown. In general sub query can be nested to any depth.

Figure: 4.46 Sub query with Multiple Levels of Nesting

CORRELATED SUBQUERY

The query ‘get suppliers name for suppliers who supply part P2’

```
SELECT SNAME
FROM S
WHERE ‘P2’ IN
            (SELECT P#
              FROM SP
              WHERE S# = S.S#)
```

The last line, the unqualified reference to S# is implicitly qualified by SP, the other reference is explicitly qualified by S. This example differs from the preceding ones in that the inner sub query cannot be evaluated once and for all the four the outer query is evaluated, because that inner sub query depends on a variable, namely S.S#, whose value changes as the system examines different rows of table S. The evaluation proceeds as follows:

a) The system examines some row of a table S, let us suppose this is the row for S1. The variable S.S# thus currently has a value S1, so the system evaluates the inner sub query

Figure: 4.47 Correlated Sub queries

SCALAR OPERATORS AND FUNCTIONS

NUMERIC OPERATORS:

DB2 supports the usual numeric operators +, -, *, and /, all with the obvious meanings. The + and – can be used with dates, times, and timestamps as well as with numbers.

CONCATENATION:

The concatenation operator || can be used to concatenate two character strings.
Ex: The expression INITIALS || LASTNAME can be used to concatenate the values of INITIALS AND LASTNAME (in that order).

CHAR

Converts a date, time, timestamp, or decimal number to its character string representation.

DATE

Converts a scalar value to a date.

Figure: 4.48 Scalar Operators and Functions

SCALAR OPERATORS AND FUNCTIONS (Cont….)

DAYS

Converts a date or timestamp to a number of days.

DECIMAL

Converts a number to decimal representation (with specified precision).

SUBSTR

Extracts a substring of a string.
Ex: The expression SUBSTR (SNAME, 1,3) extracts the first three characters of the specified supplier name.

TIMESTAMP

Converts either a single scalar value or a pair of scalar values (representing a date and time respectively), to a timestamp.

INTEGER

Converts a number to integer representation.

Figure: 4.49 Scalar Operators and Functions (Cont….)

AGGREGATE/COLUMN FUNCTIONS

AGGREGATE/COLUMN FUNCTIONS

This is also powerful in many ways; the SELECT statement as so far described is still in adequate for many practical problems. SQL therefore provides a number of special aggregate or column functions to enhance its basic retrieval power. The AGGREGATE or COLUMN functions are as follows:

COUNT	-	number of values in the column
SUM	-	sum of values in the column
AVG	-	average of the values in the column
MAX	-	largest value in the column
MIN	-	smallest value in the column

For SUM and AVG, the arguments should be numeric. In general, the argument may optionally be preceded by the key word DISTINCT to indicate redundant duplicate values.
For MAX and MIN, however DISTINCT is irrelevant and should be omitted.

Figure: 4.50 Aggregate/Column Functions

AGGREGATE/COLUMN FUNCTIONS (Cont…)

Aggregate functions has rules and restrictions as follows

1. For COUNT, DISTINCT must be specified, the special function COUNT (*)-DISTINCT not allowed – is provided to count all rows in a table without any duplicate elimination.
2. Regardless of whether DISTINCT is specified, the argument may consist of general scalar expression such as WEIGHT * 454. However, that expression cannot in turn involve in Aggregate functions.
3. Within any given query or sub query, DISTINCT can appear at most once at a given level of nesting.

For ex.

```
SELECT SUM (DISTINCT QTY),
       AVG (DISTINCT QTY),
FROM SP
…….      ;
```

This example is illegal.

Figure: 4.51 Aggregate/Column Functions (Cont…)

AGGREGATE/COLUMN FUNCTIONS (Cont…)

```
SELECT DISTINCT….
FROM…
GROUP BY.
HAVING SUM (DISTINCT..) ….      ;
```

This example is also illegal.

However the following example is legal:

```
SELECT DISTINCT ….
FROM ….
WHERE …… IN
            (SELECT DISTINCT..
              ……. );
```

4. Any nulls in the argument column are always eliminated before the function is applied; regardless of whether DISTINCT is specified, except for the case of COUNT (*), where nulls are handled just like not null value.
5. If the argument happens to be an empty set, COUNT returns a value of 0, the other functions all return null. Again, the VALUE function can be used to convert such a null in to some not null value.

Figure: 4.52 Aggregate//Column Functions (Cont…)

AGGREGATE/COLUMN FUNCTIONS (Cont…)

Ex. on AGGREGATE functions

1. Aggregate function in a SELECT clause.

The query
'Get the total number of suppliers'

```
SELECT COUNT (*)
FROM S
```

Result

5

Note that the result is still a table, but a table with just one row and unnamed column.

Figure: 4.53 Aggregate/Column Functions (Cont…)

AGGREGATE/COLUMN FUNCTIONS (Cont…)

2. Aggregate function in the SELECT clause, with DISTINCT.
The query
‘ Get the total number of suppliers currently supplying parts’

```
SELECT COUNT (DISTINCT S#)
FROM SP
```

Result

4

3. Aggregate function in the SELECT clause, with a CONDITION.
The query
‘ Get the number of shipments for part P2’

```
SELECT COUNT (*)
FROM SP
WHERE P# = ‘P2’
```

Result

4

Figure: 4.54 Aggregate/Column Functions (Cont…)

AGGREGATE/COLUMN FUNCTIONS (Cont…)

4. Aggregate function in the SELECT clause, with a CONDITION.

The query

‘ Get the total quantity of part for P2 supplied’

```
SELECT SUM (QTY)
FROM SP
WHERE P# = ‘P2’
```

Result

1000

Note:

Unless the query includes a GROUP BY or HAVING clause (at the same level of nesting), a SELECT clause that includes any AGGREGATE function references must consist entirely of such references.

```
SELECT P# SUM (QTY)
FROM SP
WHERE P# = ‘P2’;
```

Figure: 4.55 AGGREGATE/COLUMN Functions (Cont…)

AGGREGATE/COLUMN FUNCTIONS (Cont…)

AGGREGATE FUNCTION IN A SUBQUERY

The query
'Get supplier numbers for suppliers with status value less than the current maximum status value in the S table.

```
SELECT S#
FROM S
WHERE STATUS <
                    (SELECT MAX (STATUS)
                      FROM S);
```

Result

S#
S1
S2
S4

Figure 4.56 AGGREGATE/COLUMN Functions (Cont…)

AGGREGATE/COLUMN FUNCTIONS (Cont…)

AGGREGATE FUNCTION IN CORRELATED SUBQUERY

The query

'Get supplier number STATUS and CITY for all suppliers whose status is greater than or equal to the average for their particular city'.

```
SELECT S#, STATUS, CITY
FROM S SX
WHERE STATUS >=
                    (SELECT AVG (STATUS)
                      FROM S SY
                       WHERE SY.CITY = SX .CITY);
```

Result

S#	STATUS	CITY
S1	20	LONDON
S3	30	PARIS
S4	20	LONDON
S5	30	ATHENS

It is not possible to use average status for each city in this result.

Figure 4.57 AGGREGATE/COLUMN Functions (Cont…)

USE OF GROUP BY

The query
'Get the part number and the total shipment quantity for that part'.

```
SELECT P#, SUM (QTY)
FROM SP
GROUP BY P#;
```

Result

P#	
P1	600
P2	1000
P3	400
P4	500
P5	500
P6	100

Figure: 4.58 Use of Group BY

USE OF GROUP BY (Cont…)

The GROUP BY operator causes the table represented by the FROM clause to be rearranged into partitions or groups, such that within any one group all rows have the same value for the GROUP BY column.

In this Example, table SP is grouped so that one group contains all the rows for part P1, another contains all the rows for part P2, and so on. The SELECT clause is then applied to each group of the partitioned table (rather than to each row of the original table). Each expression in the SELECT clause must be single valued per group.

Ex. it can be (one of) the column(s) named in the GROUP BY clause, or a literal, or an aggregate function such as SUM that operates on all values of a given column within a group and reduces those values to a single value.

GROUP BY does not imply ORDER BY.

Figure: 4.59 Use of Group BY(Cont…)

USE OF GROUP BY (Cont…)

USE OF WHERE WITH GROUP BY

The query
'For each part supplied, get the part number and the total and maximum quantity supplied for that part, excluding shipments from supplier S1'

```
SELECT P#, SUM (QTY), MAX (QTY)
FROM SP
WHERE S# NOT EQUAL 'S1'
GROUP BY P#
```

Result

P#		
P1	300	300
P2	800	400
P4	300	300
P5	400	400

Rows that do not satisfy the WHERE clause are eliminated before any grouping is done.

Figure: 4.60 Use of Group BY(Cont…)

USE OF GROUP BY (Cont…)

USE OF HAVING

The query
'Get part numbers for all parts supplied by more than one supplier'

```
SELECT P#
FROM SP
GROUP BY P#
HAVING COUNT (*) > 1;
```

Result

P#
P1
P2
P4
P5

HAVING is to groups what WHERE is to rows, thus if HAVING is specified, GROUP BY should be specified. In other words HAVING is used to eliminate groups just as WHERE is used to eliminate rows.

Figure: 4.61 Use of Group BY(Cont…)

UNION

The UNION of two sets is the set of all elements belonging to either or both of the original sets. Since the relation is a set (set of rows), it is possible to consider the union of two relations; the result will be a set consisting of all rows appearing in either or both of the original relation. However if that result is itself to be another relation and not just a heterogeneous mixture of rows, the two original relations must be union compatible, i.e. the rows in the two relations must be 'the same shape'. In DB2 the two tables are union compatible, and the UNION operator can be applied to them if and only if:

a) They have the same number of columns, m say.
b) For all I (I = 1,2,m), the Ith column of the first table and the Ith column of the second table are compatible.

Query involving UNION:

The query
'Get part numbers for parts that either weight more than 16 pounds or are supplied by supplier SP'

```
SELECT P#
FROM P
WHERE WEIGHT > 16
UNION
SELECT P#
FROM SP
WHERE S# = 'S2';
```

Result

P1
P2
P3
P6

Figure: 4.62 Union

UNION (Cont….)

Several points arise from this simple example.

- Redundant duplicates are always eliminated from the result of a UNION, unless the UNION operator explicitly includes the ALL qualifier as given below:

For example, part P2 is selected by both of the two constituent SELECTs but it appears only once in the final result. The statement

```
SELECT P#
FROM P
WHERE WEIGHT > 16

UNION ALL

SELECT P#
FROM SP
WEHER S# = ‘S2’;
```

Will return part numbers (P1 P2 P2 (again), P3 P6).

- The primary reason for including UNION ALL in the language is that there are many situations in which a union is required and the user knows that there will not be any duplicates in the result.
- Any number of SELECTs can be UNIONed together.
- Parenthesis can be used for multiple Unions.

Figure: 4.63 Union (Cont…)

THE CATALOG

The Catalog in DB2 is a system database that contains information (descriptors) concerning various objects that are interest to DB2. Examples of such objects are Base Tables, Views, Indexes, Database, Application Plans, Packages, access privileges and so on.

A significant advantage of a relational system like DB2 is that the catalog in such a system itself consists of relations (or tables-system tables, so called to distinguish them from ordinary user tables).

In DB2 specifically, the catalog currently consists of 38 system tables. The only catalog tables we mention at this point are:

- **SYSTABLES**
- **SYSCOLUMNS**
- **SYSINDEXES**

ALIASES AND SYNONYMS:

ALIAS is an alternate name for a table (base table or view). You can define ALIAS for a table that was created by some other user and for which you would otherwise have to use a fully qualified name.

Figure 4.64 The Catalog

CATALOG (Cont.)

Ex:

CREATE ALIAS ZTEST FOR ALPHA.SAMPLE;

ALPHA's table can now be referred by simple unqualified name ZTEST.

SELECT * FROM ZTEST;

Dropping the table causes all the ALIAS related to the table will be dropped automatically.

DROP ALIAS ZTEST;

SYNONYM:

Synonym like an alias is an alternate name for the table. Difference between alias and synonym is:

Alias is private to the user who creates it, unlike synonym.
Synonym cannot refer to a remote table, unlike alias.

CREATE SYNONYM ZTEST FOR ALPHA.SAMPLE;

To Drop the Synonym

DROP SYNONYM ZTEST

Figure: 4.65 Catalog (Cont…)

CREATION OF VIEWS

A VIEW is just a description of a set of columns and rows with which we want to work.

SYNTAX

CREATE VIEW Viewname ((column (, column)….))
AS SUBQUERY
(WITH CHECK OPTION));

The sub query cannot include either UNION OR OREDER BY.

In DB2, a view that is to accept updates must be derived from a single base table.

Figure 4.66 Creation of Views

Notes:

Tables come in two kinds, base tables and views.

- A base table is a named table that is not defined in terms of other tables; in other words, it is an autonomous table one that “exists in its own right”.
- A view, by contrast, is a named table that is not autonomous and does not “exist in its own right”(although it behaves in some ways as if it did).
- To be more precise, a view is a named table that is represented, not by its own, physically separate, distinguishable stored data, but rather by its definition in terms of other named tables.

LIMITATIONS OF VIEWS

a) If a column of the view is derived from an expression involving a scalar operator or a scalar function or a literal, then INSERT operations are not allowed, and UPDATE operations are not allowed on that column, however, Delete operation are allowed.

b) If a column of the view is derived from an aggregate function, then the view is not updatable.

c) If the definition of the view involves either GROUP BY or HAVING at the outermost level, then the view is not updatable.

d) If the definition of the view involves DISTINCT at the outermost level, then the view is not updatable.

e) If the definition of the view includes a nested sub query and the FROM clause in that sub query refers to the base table on which the view is defined, then the view is not updatable.

Figure: 4.67 Limitations of Views

ADVANTAGES OF VIEWS

They provide a certain amount of logical data independence in the face of restructuring the database.

They allow the same data to be seen by different users in different ways (possible even at the same time).

The user's perception is simplified.

Automatic security is provided for hidden data.

Hidden data:

"Hidden data" refers to data not visible through some given view. Such data is clearly secure from access through that particular view. Thus, forcing users to access the database via views is a simple but effective mechanism for authorization control.

Figure: 4.68 Advantages of Views

SECURITY AND AUTHORIZATION

- **The term Security is used in the database context to mean the protection of the data in the database against unauthorized disclosure, alteration or destruction.**

- **DB2, like most other relational systems, goes far beyond pre relational systems in the degree of security it provides. The unit of data that can be individually protected ranges all the way from an entire table to a specific data value at a specific row and column position within such a table.**

- **A given user can have different access privileges on different objects for e.g. SELECT privilege only on one table, SELECT AND UPDATE privileges on another, and so on.**

GRANT AND REVOKE.

The SQL statements GRANT and REVOKE perform the following function.
First, in order to perform any operation at all on any object at all, the user must hold the appropriate privilege for the operation and object in question; otherwise, the operation will be rejected with an appropriate error or exception code.

Figure 4.69 Security and Authorization

SECURITY AND AUTHORIZATION (Cont….)

For example:
Even to execute such a simple statement as

SELECT * FROM S;

Successfully, the user must hold the SELECT privilege on table S.

DB2 recognizes a wide range of privileges.

- **Table privileges:** which have to do with operations such as SELECT that apply to tables (both base tables and views).
- **Plan and Package privileges:** which have to do with such things as the authority to use a given plan, package, or collection of packages.
- **Collection privileges:** which permit the holder to create packages in the given collection.
- **Database privileges:** which apply to operations such as the creation of a table within a particular database.
- **Use privileges:** which have to do with the use of certain “system resources”, namely storage groups, table spaces and buffer pool.

Figure 4.70 Security and Authorization (Cont…)

SECURITY AND AUTHORIZATION (Cont….)

The system administration privilege (SYSADM) is shorthand for the collection of all other privileges in the system. Thus a user holding the SYSADM privilege can perform any operation in the entire system, providing it is legal.

This gives how The Overall Security Mechanism Works:

When DB2 is first installed, part of the installation process involves the designation of a specially privileged user as the system administrator for that DB2 system. (The system administrator is identified to DB2 by an authorization ID, of course, just like everyone else) That user, who is automatically given the SYSADM privilege, will be responsible for overall control of the system throughout the system lifetime.

GRANT
Granting privileges is done by means of the GRANT statement. The general format of that statement is:

GRANT privileges (ON (type) objects) TO users

Where “privileges” is a list of one or more privileges separated by commas or the phrase ALL PRIVILEGES, or a special keyword PUBLIC (meaning all users).

Figure 4.71 Security and Authorization (Cont…)

SECURITY AND AUTHORIZATION (Cont….)

The ON clause does not apply when the privileges being granted are system privileges.

Examples:
GRANT SELECT ON TABLE S TO CHARLEY;

GRANT SELECT, UPDATE (STATUS, CITY) ON TABLE S TO JUDY, JACK, JOHN;

GRANT ALL PRIVILEGE ON TABLE S, P TO PHIL, FRED;

GRANT SELECT ON TABLE P TO PUBLIC;

GRANT SELECT ON TABLE P TO PUBLIC;

GRANT DELETE ON S TO PHIL;

Package and plan privileges:
GRANT EXECUTE ON PLAN PLANB TO JOHN;

Collection privileges:
GRANT CREATE IN COLLQ TO TOMSON;

Database Privileges:
GRANT CREATETAB ON DATABASE DBX TO NANCY;

Figure 4.72 Security and Authorization (Cont…)

SECURITY AND AUTHORIZATION (Cont….)

REVOKE

If user U1 grants some privilege to some other user U2, user U1 can subsequently revoke that privilege from user U2.
Revoking privileges is done by means of the REVOKE statement, whose general format is very similar to that of the GRANT statement.

REVOKE privileges (ON (type) objects) FROM users;

Examples:

REVOKE SELECT ON TABLE S FROM CHARLEY;

REVOKE UPDATE ON TABLE S FROM JOHN;

REVOKE CREATETAB ON DATABSE DBX FROM NANCY;

Figure 4.73 Security and Authorization (Cont…)

INTEGRITY

The term “integrity” is used in database contexts to refer to the accuracy, validity, or correctness of the data in the database.
Maintaining integrity is of paramount importance. It can be handled by the system rather than by the user. In order that it may carry out this task, the system needs to be aware of any integrity constraints or rules that apply to the data.

The relational model, by contrast, includes two general integrity rules Primary key, Foreign keys

Primary key:

The Primary key of a table is just a unique identifier for that table.
Entity Integrity
The first of the two general integrity rules of the relational model is called the entity integrity rule

- No component of the primary key of a base table is allowed to accept nulls.

Foreign key:

A foreign key value represents a reference to the row containing the matching primary key value (the referenced row or target row). The problem of ensuring that the database does not contain any invalid foreign key values is therefore known as the referential integrity problem

Figure 4.74 Integrity

INTEGRITY(Cont…)

The constraint that values of a given foreign key must match values of the corresponding primary key is known as a ‘Referential Constraint.’

The table that contains the foreign key is known as the referencing table and the table that contains the corresponding primary key is known as the referenced table or target table.

Figure: 4.75 Integrity (Cont…)