Unit 2

DB2 and SQL

Outline of Unit 2

2.1 Overview

2.2 Data Definition

2.3 Data Manipulation

2.4 The System Catalog

2.5 Embedded SQL

2.1 Overview

Background

- Relational Model: proposed by Codd, 1970

Ref: CACM Vol. 13, No.6, "A relational model of data for large shared data banks"

	System R	INGRES
Developer	IBM San Jose Res. Lab 1974 - 1979	UC Berkeley late 1970 - early 1980
Machine	IBM System 370	DEC PDP
O. S.	VM / CMS	UNIX
Query Language	SQL	QUEL
Language Embedded	COBOL or PL/1	COBOL, PASCAL, C FORTRAN, BASIC
Commercial Product	DB2, SQL / DS	Commercial INGRES
Distributed OB	R*	Distributed INGRES
OO Extension	Starburst	POSTGRES

Relational Databases

- Definition: A ***Relational Database*** is a database that is perceived by its users as a collection of tables (and nothing but tables).

<e.g.> Supplier-and-Parts Database

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

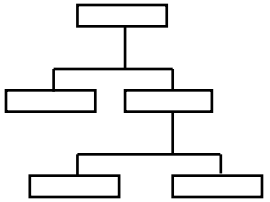
P

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	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

(Hierarchical)
IMS



Relational Databases (cont.)

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	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

- S, P, SP: 3 relations (tables)
- A row in a relation is called a tuple (record)
- S, P: entities; SP: relationship
- primary key: S# in S, P# in P, (S#, P#) in SP
- atomic: not a set of values, instead of repeating group

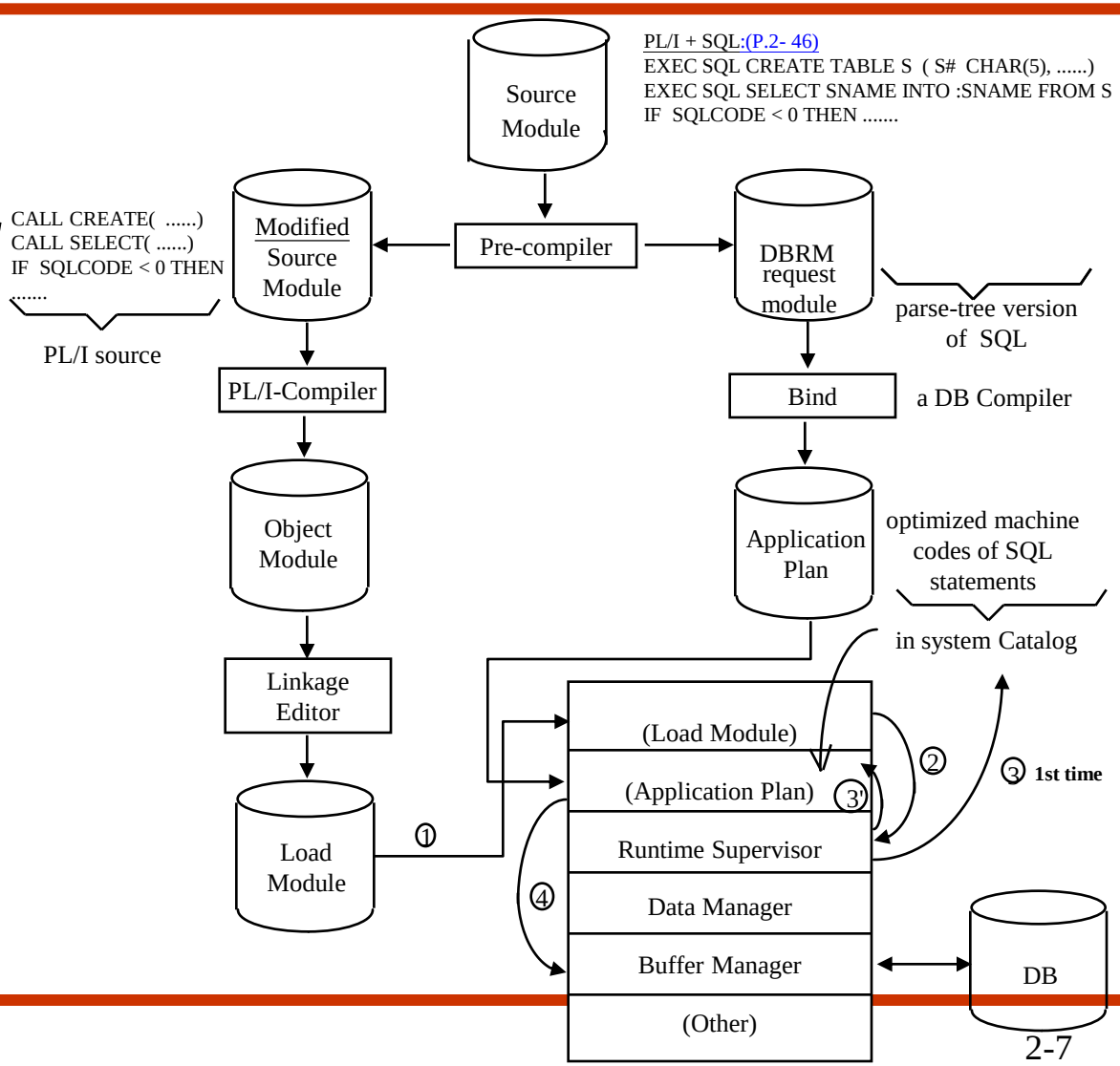
< e.g. >

S#	P#
S1	{ P1, P2, P3, P4, P5, P6 }
S2	{ P1, P2 }
.	.
:	:

⇨ atomic Normalization

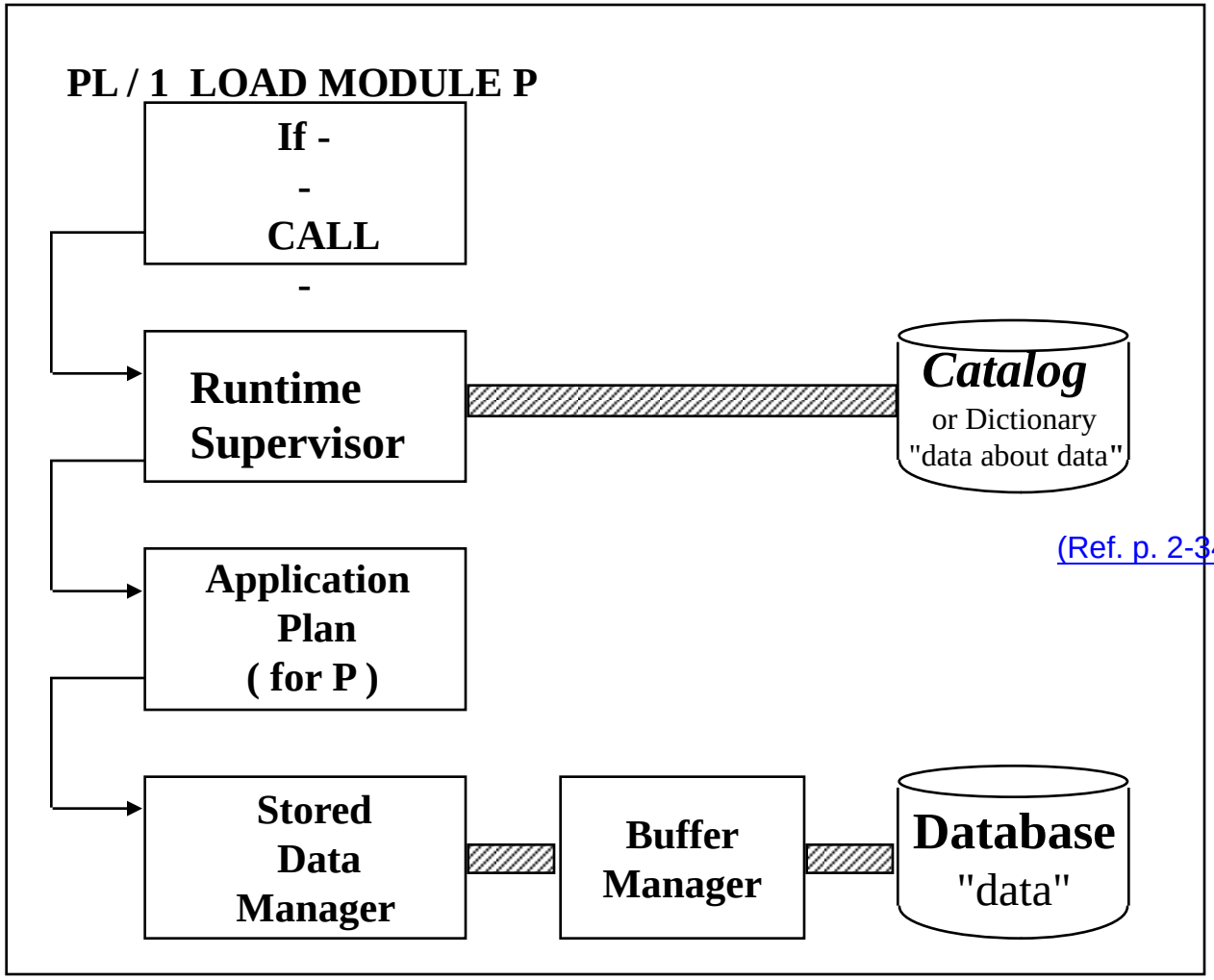
Major System Components: DB2

1. ***Pre-compiled***
2. ***Bind***
3. ***Runtime Supervisor***
4. ***Data Manager***
5. ***Buffer Manager***



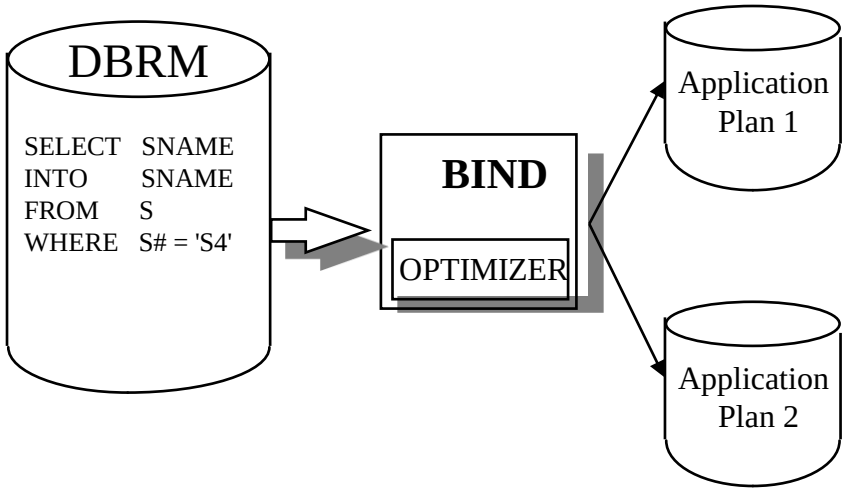
Major System Components: Execution time

Execution time



(Ref. p. 2-34)

Major System Components: Optimizer



- Plan 1 **(without index): SCAN S**

  if S # = 'S4' then extract name field

  go to SCAN S

  ⋮

- Plan 2 **(with sorted index): Binary Search X**

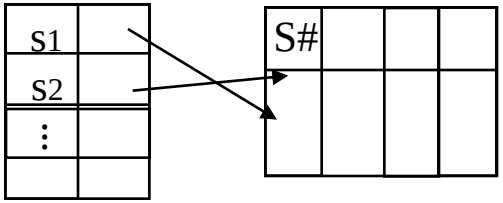
  if **X**.key = 'S4' then .....

Be chosen by optimizer (Ref. p. 2-34)

- Considerations :
  1. Which table ?
  2. How big ?
  3. What index exist ?

  ⋮

X: S#_index



2.2 Data Definition

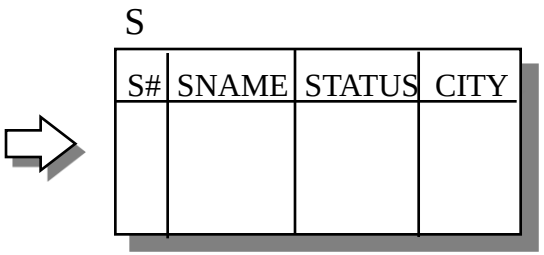
DDL of SQL

- **Base Table**
  - **Create Table**
  - **Drop Table**
  - **Alter Table**
- **Index**
  - **Create Index**
  - **Clustering Index**
  - **Drop Index**
- **View**
  - **Create View**
  - **Drop View**

Base Tables

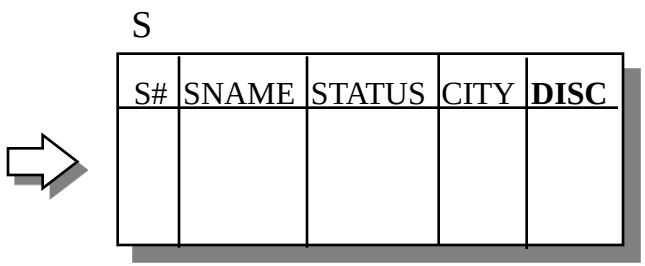
- **A named table**

<e.g. 1> **CREATE** TABLE **S**
(S# CHAR (5) NOT NULL,
SNAME CHAR (20),
STATUS SMALLINT,
CITY CHAR (15));



- **Data can be entered by**
  - (1) **INSERT** statement
  - (2) DB2 load utility

<e.g. 2> **ALTER** TABLE **S**
ADD **DISC** SMALLINT;



<e.g. 3> **DROP S ;**

- Description of S is removed from **system catalog**.
- All index and views defined on S are automatically dropped.

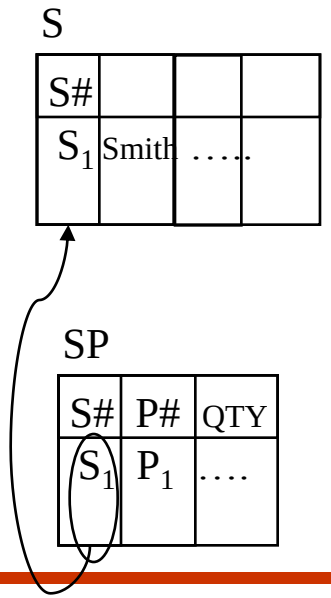
Base Tables (cont.)

- **FOREIGN KEY**

  FOREIGN KEY (*column-commalist*)
  REFERENCES *base-table* [(*column-commalist*)]
  [ON DELETE *option*]
  [ON UPDATE *option*]

- **CHECK**

  CHECK (*conditional-expression*)



<e.g.> : **CREATE TABLE SP** (Ref. p. 3-9)
(S# S# NOT NULL, P# P# NOT NULL, QTY QTY NOT NULL,
PRIMARY KEY (S#, P#) ,
FOREIGN KEY (S#) REFERENCES S
ON DELETE CASCADE
ON UPDATE CASCADE ,
FOREIGN KEY (P#) REFERENCES P
ON DELETE CASCADE
ON UPDATE CASCADE ,
CHECK (QTY >0 AND QTY <5001)) ; (Ref. p. 10-16, Integrity Rule)

Indexes

<e.g.1>: CREATE INDEX **X** ON **SP** (P# ASC, S# ASC);

<e.g.2> : CREATE UNIQUE INDEX **XS** ON **SP** (S#, P#)

- enforced that no two tuples have the same index field.

<e.g.3>: DROP INDEX **X**;

- Definition of **X** is removed from **Catalog**.

X

P#	S#	Pointer
P1	S1	..
P1	S2	..
P2	S1	..
P2	S2	..
P2	S3	..
P2	S4	..
P3	S1	..
P4	S1	..
P4	S4	..
P5	S1	..
P5	S4	..
P6	S1	..

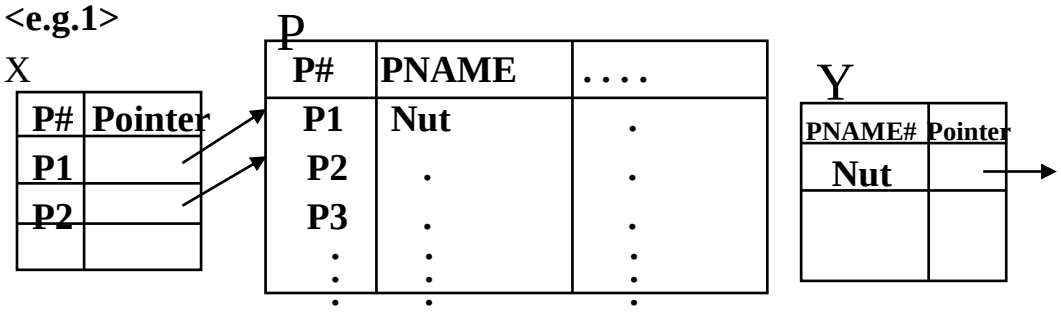
SP

S#	P#	QTY	PRICE
S1	P1	300	50
S1	P2	200	30
S1	P3	400	70
S1	P4	200	100
S1	P5	100	90
S1	P6	100	40
S2	P1	300	25
S2	P2	400	75
S3	P2	200	48
S4	P2	200	79
S4	P4	300	65
S4	P5	400	55

Clustering index

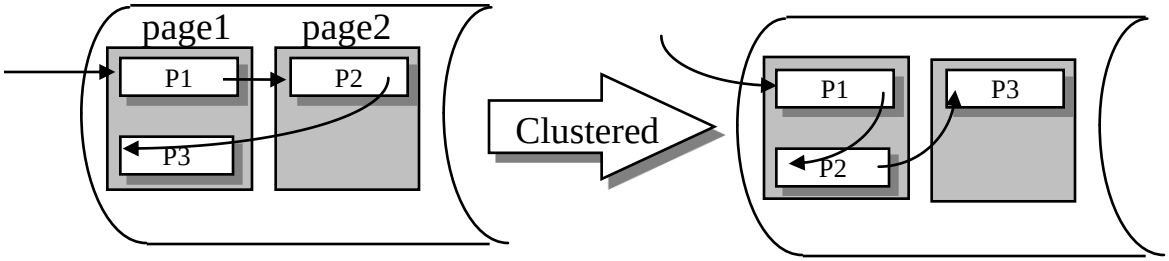
- Logical sequence ≅ Physical sequence

<e.g.1>



logical sequence : P1 < P2 < P3 < . . .

physical sequence :

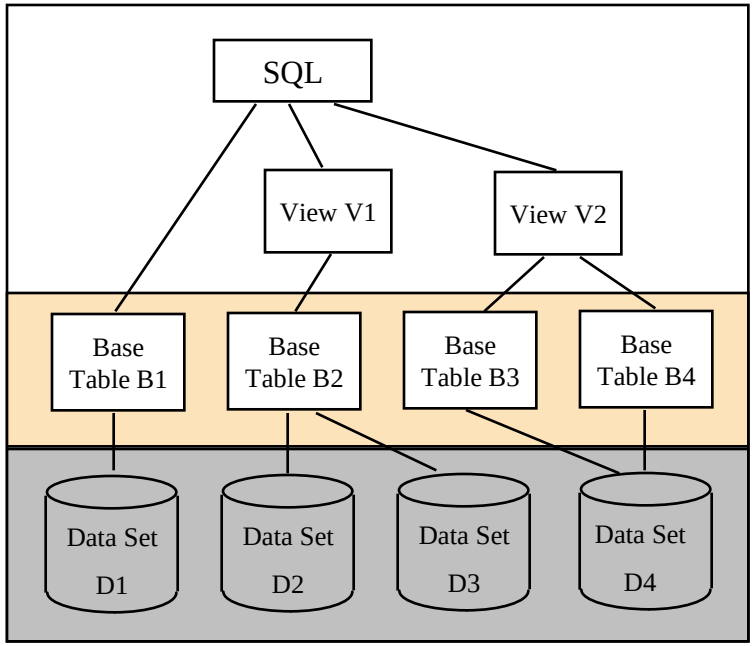


<e.g.2> CREATE INDEX **X** ON **P** (P#) CLUSTER;

Note: A given base table can have at most one cluster index.

Views

- Virtual table (doesn't really exist)
- No stored file
- Definition of view is stored in **system catalog**
- A base table may be stored in several files
- A file may contain several base tables
- A view may be derived from several base tables
- A base table may derive several views



Views: Example

view name

<e.g.> **CREATE VIEW LONDON-SUPPLIERS**

AS **SELECT S# , SNAME , STATUS**

FROM S
WHERE CITY = ' LONDON' ?

view definition in catalog

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

LONDON-SUPPLIERS

S#	SNAME	STATUS
S1	**Smith**	**20**
S4	**Clark**	**20**

Views: Example (cont.)

Can be used as base table: e.g. S, P, SP

<e.g.>

```
SELECT *
FROM  LONDON-SUPPLIERS
WHERE STATUS <  50
```

converted by Bind (ref. p. 2-7)

```
SELECT S# , SNAME , STATUS
FROM    S
WHERE STATUS < 50
AND  CITY  = ' LONDON'
```

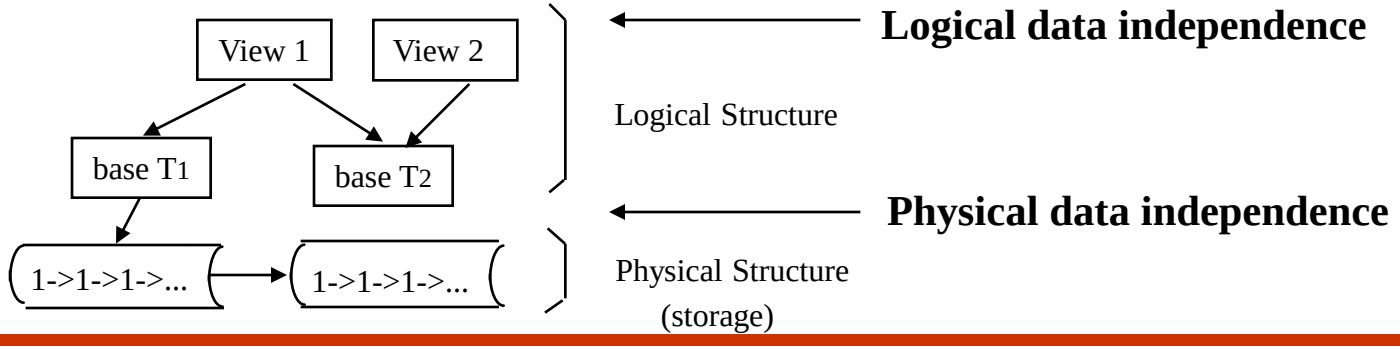
Views: Advantages

- **Advantages of views**:
  - (1) **Provide logical data independence :**

    Logical data independence (e.g. Relation): users and user programs are not dependent on logical structure of the database.

    Two aspects of logical data independence : Growth and restructuring.

    (v.s. **Physical data independence** (e.g. B-tree): users and user programs are not dependent on physical structure of the stored database.)
  - (2) **Allow same data to be seen by different users in different ways.**
  - (3) **User perception is simplified.**
  - (4) **Automatic security is provided for hidden data**



2.3 Data Manipulation

DML of SQL

- **Retrieval Operation**
  - SELECT
- **Update Operation**
  - UPDATE
  - DELETE
  - INSERT
- **Expressions**
  - Table Expressions
- **Operations on View**

Retrieval Operations

- **Get color and city for "non-Paris" parts with weight greater than ten.**

```
SELECT  P.COLOR,  P.CITY
FROM    P
WHERE  P.CITY < >  'Paris'
AND      P.WEIGHT > 10;
```

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

- **DISTINCT**

```
SELECT  DISTINCT  P.COLOR,  P.CITY
FROM    P
WHERE  P.CITY < >  'Paris'
AND      P.WEIGHT  >  10;
```

P#	COLOR	CITY
P1	Red	London
P3	Blue	Rome
P4	Red	London
P6	Red	London

- **ORDER BY**

```
SELECT  DISTINCT  P.COLOR,  P.CITY
FROM    P
WHERE  P.CITY  < >  'Paris'
AND      P.WEIGHT  >  10
ORDER   BY  CITY  DESC;
```

P#	COLOR	CITY
P3	Blue	Rome
P1	Red	London
P4	Red	London
P6	Red	London

Retrieval Operations (cont.)

- For all parts, get the part number and the weight of that part in grams.

```
SELECT  P.P#,  P.WEIGHT * 454  AS  GMWT
FROM    P ;
```

  – **If the AS MWT specification had been omitted, the corresponding result column would effectively have been unnamed.**

- Get full details of all suppliers.

```
SELECT  *    – or "SELECT  S.*"  (i.e., the "*" can be qualified )
FROM    S  ;
```

- Get the total number of suppliers.

```
SELECT  COUNT ( * )  AS  N
FROM    S ;
```

Retrieval Operations (cont.)

- Get the maximum and minimum quantity for part P2.

```
SELECT  MAX (SP.QTY) AS MAXQ,
        MIN (SP.QTY) AS MINQ
FROM    SP
WHERE  SP. P#  =  'P2';
```

- For each part supplied, get the part number and the total shipment quantity.

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

Retrieval Operations (cont.)

- Get part numbers for all parts supplied by more than one supplier.

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

- Get supplier names for suppliers who supply part P2.

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

Update Operations

- Single-row INSERT.

```
INSERT
INTO     P(P#, PNAME, COLOR, WEIGHT, CITY)
VALUES  ('P8', 'Sprocket', 'Pink', 14, 'Nice');
```

- Multi-row INSERT.

```
INSERT
INTO     TEMP (S#, CITY)
         SELECT  S.S#,  S.CITY
         FROM    S
         WHERE  S.STATUS > 15  ;
```

- Multi-row UPDATE.

```
UPDATE  P
SET       COLOR  =  'Yellow' ,
          WEIGHT  =  P.WEIGHT  +  5
WHERE   P.CITY  =  'Paris';
```

Update Operations (cont.)

- Multi-row UPDATE.

```
UPDATE  P
SET     CITY = ( SELECT  S.CITY
                 FROM    S
                 WHERE   S.S#  = 'S5')
WHERE   P.COLOR = ' Red '
```

- Multi-row DELETE.

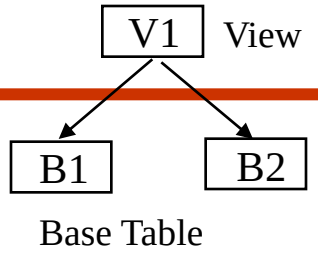
```
DELETE
FROM      SP
WHERE     'London‘ = (SELECT S.CITY
                      FROM S
                      WHERE S.S# = SP.S#);
```

Table Expressions

- The SELECT, FROM, WHERE, GROUP BY, and HAVING clause.
- A Comprehensive Example

```
SELECT  P. P#, 'Weight  in grams = ' AS TEXT1,
              P.WEIGHT * 454  AS  GMWT,
              P.COLOR,   'Max  quantity = ' AS TEXT2,
             MAX (SP.QTY) AS MQY
 FROM     P, SP
WHERE    P.P#  =  SP.P#
AND      ( P.COLOR = 'Red‘ OR  P.COLOR = ' Blue')
AND       SP.QTY > 200
GROUP    BY  P. P#, P.WEIGHT, P.COLOR
HAVING  SUM  (SP.QTY) > 350;
```

DML operations on View



- Retrieval (SELECT): no problem
- Update (INSERT, DELETE, UPDATE): ?

(1) Column Subset: **theoretically updateable if contains primary key.**

<e.g.1> :

```
CREAT VIEW  S#_CITY
AS  SELECT  S# ,  CITY
FROM   S;
```

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

Base Table

S#_CITY

S#	CITY
S1	London
S2	Paris
S3	Paris
S4	London
S5	Athens

View

```
INSERT
INTO S#_CITY
VALUES ('S6', 'Rome');
```

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
S6	*Null*	*Null*	*Rome*

DML on View: Column Subset without key

<e.g.2>

```
CREATE  VIEW STATUS_CITY
AS   SELECT STATUS, CITY
FROM  S;
```

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

STATUS_CITY

STATUS	CITY
20	London
10	Paris
30	Paris
30	Athens

```
INSERT
INTO  STATUS_CITY
VALUES (30, 'Rome')
```

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
Null	*Null*	*30*	*Rome*

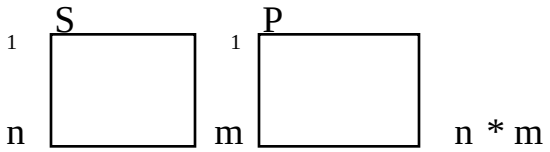
Primary key cannot be null !!

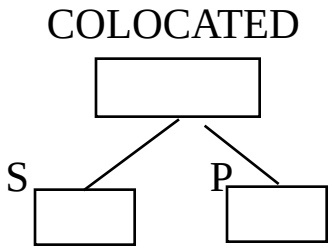
DML on View: Row Subset and Join

(2) Row Subset: *updateable !*

(3) Join: some are not updateable.

- **CREATE VIEW COLOCATED (S#, SNAME, S.CITY, P#, PNAME, P.CITY)**
  AS SELECT S# , SNAME , S.C ITY, P# , PNAME , P.CITY
  FROM S, P
  WHERE S.CITY=P.CITY;





- If issued the following command:

  UPDATE COLOCATED

  SET S.CITY = 'Athens‘

  WHERE S.CITY ='London'

- Then S.CITY ≠ P.CITY

  ⇧

  Violate the definition of the view!!

COLOCATED

S#	SNAME	S.CITY	P#	PNAME	P.CITY
S1	Smith	London	P1	Nut	London
S1	Smith	London	P4	Screw	London
S1	Smith	London	P6	Cog	London
S2	Jones	Paris	P2	Bolt	Paris
S2	Jones	Paris	P5	Cam	Paris
S3	Blake	Paris	P2	Bolt	Paris
S3	Blake	Paris	P5	Cam	Paris
S4	Clark	London	P1	Nut	London
S4	Clark	London	P4	Screw	London
S4	Clark	London	P6	Cog	London

DML on View: Statistical Summary

(4) Statistical Summary : not updateable.

<e.g.> : **CREATE VIEW PQ(P# , TOTQTY)**
AS SELECT P# , SUM(QTY)
FROM SP
GROUP BY P# ;

No stored data item for the field "TOTQTY"

P#	TOTQTY
P1	600
P2	1000
:	:

PQ

P#	TOTQTY
P1	600

View

SP

S#	P#	QTY

Base

SP

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

2.4 The System Catalog

System Catalog: Concept

- **The Catalog Structure**
  - SYSTABLES
  - SYSCOLUMNS
  - SYSINDEXES

	cc	record	creator	...
S	4	5	Yang	
P	5	6	Yang	
SP	3	12	Yang	

- **System Catalog:** ***"Data about data"*** (Ref. p. 2-8)
  i.e. information of base tables, view, indexes, users, and access privileges that are of interest to the system itself.

- **Optimizer**: use index info. to choose access method. (Ref. p. 2-9)

- **Authorization subsystem**: use access privilege to grant or deny user requests.

	T1	T2	...
u_1	R	R	
u_2	W	R	
:			
u_n	O	W	

access control matrix

- ***Querying the catalog:*** by SQL DML

Updating the Catalog

- Cannot use the SQL update, delete and insert, because it would be too easy to destroy information!
- It is data definition statements (i.e. CREATE, DROP, ALTER) that perform such updates.
  - CREATE = INSERT into catalog
  - DROP = DELETE from catalog
  - ALTER = UPDATE catalog

Updating the Catalog: Example

- <e.g.>: CREATE TABLE S
  (S# CHAR(5) Not Null,
  SNAME CHAR(20) ,
  STATUS SMALLINT,
  CITY CHAR(5);

SYSTABLE

NAME	CREATOR	
...		
. . .	. . .	...
. . .	. . .	...
. . .	. . .	...
S		

SYSCOLUMNS

NAME	TBNAME	...
. . .	. . .	
. . .	. . .	
. . .	. . .	
S#	S	
SNAME	S	
STATUS	S	
CITY	S	

Updating the Catalog: COMMENT

- Catalog also includes entries for catalog tables.

SYSTABLE

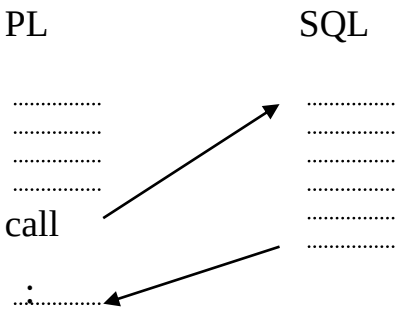
NAME	CREATOR	. . .	REMARK
SYSTABLE	SYSIBM	. . .	. . .
SYSCOLUMN	SYSIBM	. . .	. . .
.	.	. . .	. . .
.	.	. . .	. . .
S	Yang	. . .	Supplier
P	Yang	. . .	Part
SP	Yang	. . .	. . .

- The only statement that updates catalog: *COMMENT*
  - <e.g.>: COMMENT ON TABLE S IS Supplier

2.5 Embedded SQL

Embedded SQL: Dual-mode

- Dual-mode principle: any SQL statement that can be used at terminal (interactive), can also be used in an application program (programmable).



- **PL/I** (Record operations) vs. **SQL** (Set operations)

Embedded SQL: a Fragment

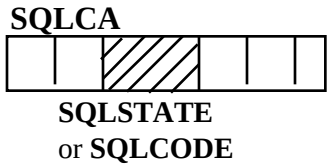
- **<e.g.> Fragment of a PL/I program with embedded SQL**

```
 1  EXEC  SQL  BEGIN  DECLARE  SECTION ;
 2      DCL  SQLSTATE  CHAR(5) ;
 3      DCL  P#            CHAR(6) ;
 4      DCL  WEIGHT     FIXED  DECIMAL(3) ;
 5  EXEC  SQL  END  DECLARE  SECTION ;
 6  P#  = ' P2 '  ;                          /*  for example                */
 7  EXEC  SQL  SELECT  P.WEIGHT
 8             INTO     :WEIGHT
 9             FROM    P
10             WHERE  P. P#  =  :P# ;
11  IF  SQLSTATE  = ' 00000 '
12  THEN  ….  ;                          /*  WEIGHT = retrieved value  */
13  ELSE  ….  ;                          /*  some exception occurred     */
```

Embedded SQL: a Fragment (cont.)

1. Embedded SQL statements are prefix by EXEC SQL.
2. Executable statements can appear wherever.
   (non-executable statements: e.g. DECLARE TABLE, DECLARE CURSOR).
3. SQL statements can reference host variable. (PL/I var., :City)
4. Any table used should be declared by DECLARE TABLE, because it is used by pre-compiler.
5. **SQLSTATE/SQLCODE**: feedback information of SQL, stored in **SQLCA** (**SQL C**ommunication **A**rea).

SQLSTATE	**= 0**	**success**
	> 0	**warning**
	< 0	**error**



6. Host variables must have compatible data type with SQL field.
7. Host variables can have same name as database fields.
   e.g. **City**, **:City**
   (SQL) (PL/I)

Operation: Singleton SELECT

- Singleton SELECT:

```
EXEC SQL SELECT STATUS
INTO     :RANK
FROM     S
WHERE    S#=: GIVENS#;
```

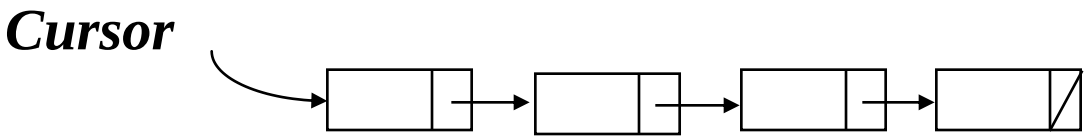
  - If only one record is satisfied: SQLCODE = 0
  - If no record is satisfied: SQLCODE > 0
  - If more than one record are satisfied: SQLCODE < 0
  - How to deal with NULL value? Indicator variable!

```
EXEC SQL SELECT STATUS
INTO     :RANK :RANKIND
FROM     S
WHERE    S#=:GIVENS#
```

    - RANKIND: an indicator variable, 15-bit signed binary integer.
    - If RANKIND = -1 THEN ……/* Status was NULL */

Operation: Multiple SELECT

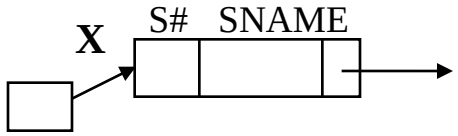
- Multiple SELECT:
  - How to handle the cases that more than one record are satisfied?



Cursor

- A kind of pointer that can be run through a set of records.

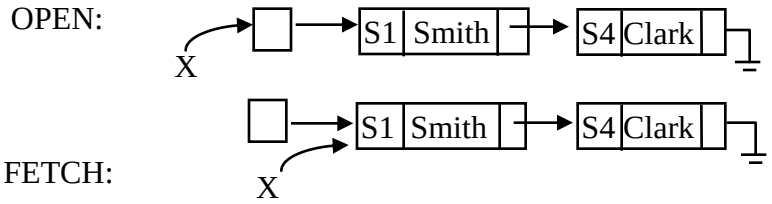
```
                EXEC SQL DECLARE X CURSOR FOR  /*define cursor S*/
define cursor   SELECT  S#, SNAME
                FROM    S
                WHERE   CITY =:Y;
```

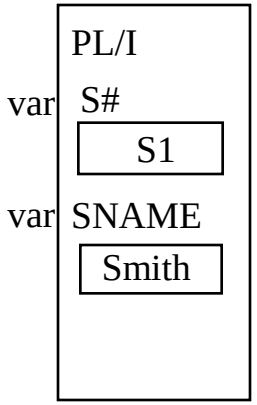



```
open cursor     EXEC SQL OPEN  X; /*activate cursor, execute the query*/
                DO for all S records accessible via X;
                    EXEC SQL FETCH X INTO :S#, :SNAME
                     .....................  /*advance pt., assign values */
                END;

close cursor    EXEC SQL CLOSE   X;  /*deactivate cursor X*/
```

e.g. Y = 'London'





Embedded SQL: An Example

- Embedded SQL

A comprehensive example

The program accepts four input values : a part number (GIVENP#), a city name (GIVENCIT), a status increment (GIVENINC), and a status level (GIVENLVL). The program scans all suppliers of the part identified by GIVENP#. For each such supplier, if the supplier city is GIVENCIT, then the status is increased by GIVENINC; otherwise, if the status is less than GIVENLVL, the supplier is deleted, together with all shipments for that supplier. In all cases supplier information is listed on the printer, with an indication of how that particular supplier was handled by the program.

Embedded SQL: An Example (cont.)

```
SQLEX: PROC  OPTIONS  (MAIN)  ;

        DCL  GIVENP #          CHAR(6)   ;
        DCL  GIVENCIT          CHAR(15)  ;
        DCL  GIVENINC          FIXED  BINARY(15)  ;
        DCL  GIVENLVL          FIXED  BINARY(15)  ;
        DCL  S#                CHAR(5)  ;
        DCL  SNAME             CHAR(20)  ;
        DCL  STATUS            FIXED  BINARY(15)  ;
        DCL  CITY              CHAR(15)  ;
        DCL  DISP              CHAR(7)  ;
        DCL  MORE_SUPPLIERS  BIT(1)  ;

        EXEC  SQL  INCLUDE  SQLCA  ;  /* p.2-41 */

        EXEC  SQL  DECLARE  S  TABLE
                          (  S#        CHAR(5)     NOT  NULL,
                             SNAME     CHAR(20)    NOT  NULL,
                             STATUS    SMALLINT    NOT  NULL,
                             CITY      CHAR(20)    NOT  NULL ) ;

            EXEC  SQL  DECLARE  SP  TABLE
                     (  S#        CHAR(5)     NOT  NULL,
                        P#        CHAR(6)     NOT  NULL,
                        QTY       INTEGER     NOT  NULL ) ;
```

PL/I Var.

Back 2-7

Embedded SQL: An Example (cont.)

```
EXEC SQL DECLARE Z CURSOR FOR
         SELECT S#, SNAME, STATUS, CITY
         FROM   S
         WHERE  EXISTS
                ( SELECT *
                  FROM   SP
                  WHERE  SP.S# = S.S#
                  AND    SP.P# = :GIVENP# )     i.e. P2
                FOR UPDATE OF STATUS ;

EXEC SQL WHENEVER NOT FOUND CONTINUE ;
EXEC SQL WHENEVER SQLERROR CONTINUE ;
EXEC SQL WHENEVER SQLWARNING CONTINUE ;

ON CONDITION ( DBEXCEPTION )
BEGIN ;
    PUT SKIP LIST ( SQLCA ) ;
    EXEC SQL ROLLBACK ;
    GO TO QUIT ;
END ;
```

Embedded SQL: An Example (cont.)

Main → **GET LIST (GIVENP#, GIVENCIT, GIVENINC, GIVENLVL) ;**

```
EXEC SQL OPEN Z ;
IF SQLCODE < > 0   /* abnormal */
THEN SIGNAL CONDITION ( DBEXCEPTION ) ;
```

SQLCODE =0 → **MORE_SUPPLIERS = ' 1' B ;**

```
DO WHILE ( MORE_SUPPLIERS ) ;
    EXEC SQL FETCH Z INTO :S#, :SNAME, :STATUS, :CITY ;
    SELECT ;  /* case */   /* a PL/I SELECT, not a SQL SELECT */
    WHEN ( SQLCODE = 100 )    /* Not found */
        MORE_SUPPLIERS = ' 0 ' B ;
    WHEN ( SQLCODE < > 100 & SQLCODE < > 0 )   /* Warning */
        SIGNAL CONDITION ( DBEXCEPTION ) ;
```

Embedded SQL: An Example (cont.)

```
WHEN ( SQLCODE = 0 )   /* success */
            DO ;
                DISP = ' bbbbbbb ' ;    /*  empty the display buffer */
                IF CITY = GIVENCIT
                THEN
                     DO ;
                        EXEC SQL UPDATE S
                                    SET STATUS = STATUS + : GIVENINC;
                                    WHERE   CURRENT OF Z ;
                        IF SQLCODE < > 0
                        THEN SIGNAL CONDITION ( DBEXCEPTION ) ;
                        DISP = ' UPDATED ' ;
                     END ;
                ELSE
                     IF STATUS < GIVENLVL
                     THEN
                          DO ;
                             EXEC SQL DELETE
                                      FROM    SP
                                      WHERE   S# = : S# ;
```

Embedded SQL: An Example (cont.)

```
                    IF  SQLCODE  < >  0  &  SQLCODE < > 100
                    THEN  SIGNAL  CONDITION ( DBEXCEPTION );
                     EXEC  SQL  DELETE
                                FROM     S
                                WHERE   CURRENT  OF  Z  ;
                    IF  SQLCODE < > 0
                    THEN  SIGNAL  CONDITION  ( DBEXCEPTION);
                    DISP  = 'DELETED '   ;
                END ;
            PUT  SKIP  LIST  ( S#, SNAME, STATUS, CITY, DISP ) ;
        END  ;   /*  WHEN  ( SQLCODE = 0 )  */
    END  ;   /* PL/I  SELECT  */
  END  ;    /* DO  WHILE */
  EXEC  SQL  CLOSE  Z  ;
  EXEC  SQL  COMMIT  ;      /* normal termination */
QUIT: RETURN  ;
END  ;   /*  SQLEX  */
```

Program Exercise 1: Using DBMS

- Due Date:
- EX. 4.1-4.6 **(p.99-100)**

  Using the suppliers-parts-projects database, write a program with embedded SQL statements to list all supplier rows, in supplier number order. Each supplier row should be immediately followed in the listing by all project rows for projects supplied by that supplier, in project number order.

  - **create database**
  - **selection**
  - **update**
  - **query catalog**
    .
    .
    .
  - **embedded SQL (program)**