DB2 & SQL

* SQL（Structured Query Language）

SYSTEM	LANGUAGE
IBM DB2	SQL
INGRES	QUEL

* DB2：A relational DBMS run on IBM system based on IBM system R project.
* SQL：the relational database language used in DB2.

* All data values are atomic: at every row and column position in every table, there is always exactly one data value, never a set of values.

  - Do not allow represent groups.
  - Explicit data values: no links or pointer that connect one table to another.
  - Relational tables are at the external and conceptual levels.

* How primary keys, foreign keys and constraints are (or are not) implemented in SQL?

* DDL and DML
  - Interactive query language (on-line) and a database programming language (embedded in applications).

```
Select CITY
From S
Where S# = 'S4' ;
```

CITY
London

(a) 互動式的 (DB2)

```
EXEC SQL Select CITY
         Into : XCITY
         From S
         Where S# = 'S4' ;
```

XCITY
London

(b) 內嵌於 COBOL or FORTRAN。

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

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

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

Fig. The suppliers-and-parts database

S#	P#
S2	P1
S2	P2
S4	P2
S4	P4
S4	P5

Instead of

S#	P#
S2	{P1, P2}
S4	{P2, P4, P5}

Fig. Atomic values

* A base table is a "real" table: a table that physically exists. (conceptual record type)
* A view is a "virtual" table, a table that does not directly exist in physical storage, but looks to the user as if it did.

- Derived from one or more base tables, or from other views.
- Only definitions are stored.

* A table as seen by the user can be a base table or it can be a view.
* Several base tables can share the same file, and one base table can be spread over several files.

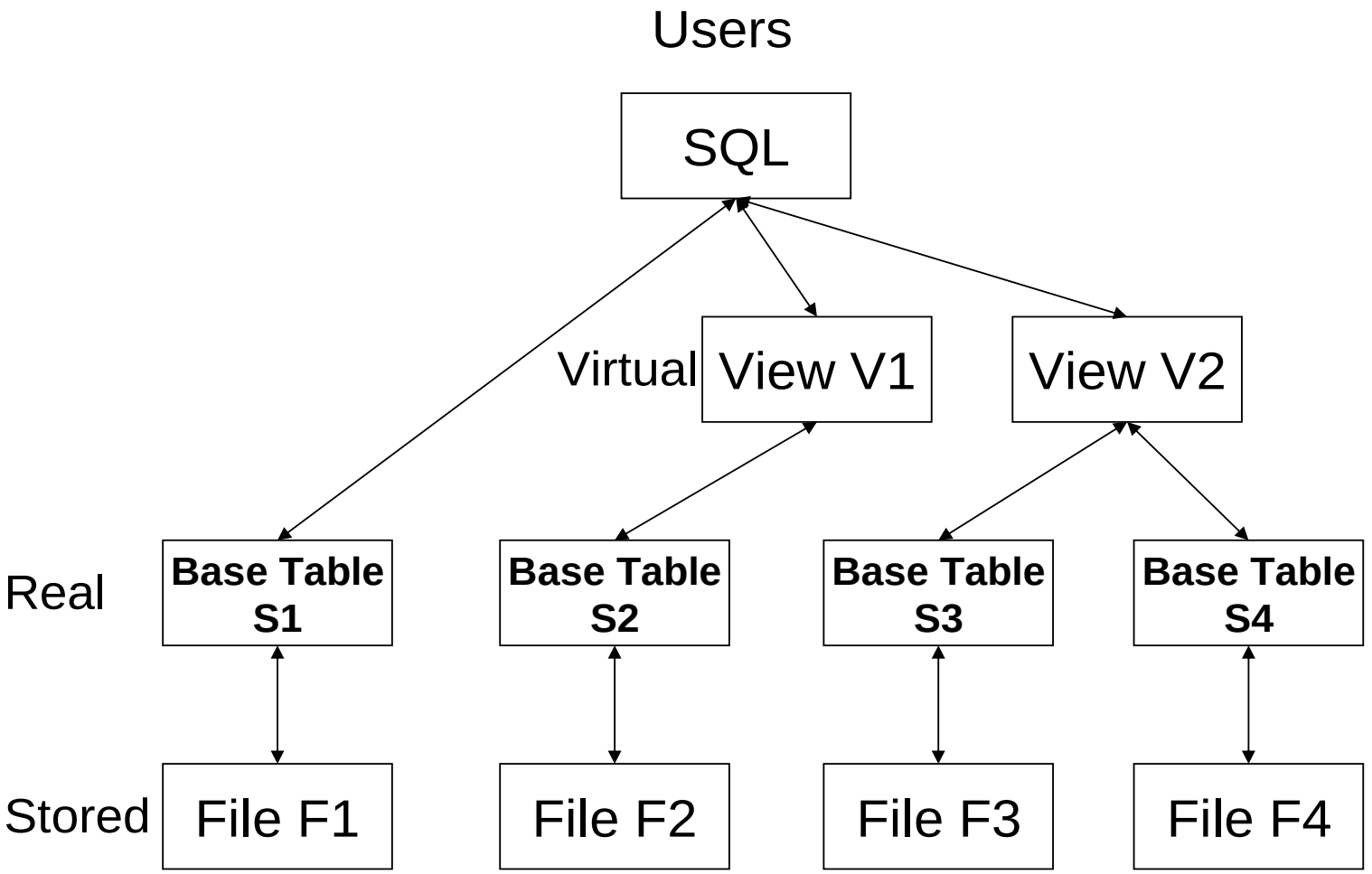

Fig. User perception of the system （DB2）

DDL（Data Definition Language）

* Create Table, View, Index
  Drop Table, Vies, Index
  Alter Table：An existing base table can be altered at any time by the addition of a new column at the right.

* Define objects at the external level (views), and the conceptual level (base table), and even the internal level (index).

* A table in a relational system consists of a row of column heading, together with zero or more rows of data values.

* No row ordering.
* Columns are ordered. (different from the relational data model)
* Syntax：

[ ]：optional

(..)：repeated

- CREATE TABLE base-table

(column-definition [, column-definition] ...

[, primary-key-definition]

[, foreign-key-definition [, foreign-key-definition] ...]);

Column definition: column data-type [NOT NULL]

```
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#));

CREATE TABLE SP
        ( S#          CHAR(5)        NOT NULL,
          P#          CHAR(6)        NOT NULL,
          QTY         INTEGER        NOT NULL,
          PRIMARY KEY (S#, P#),
          FOREIGN   KEY  (S#) REFERENCE S,
          FOREIGN   KEY  (P#) REFERENCE P)
```

* The primary key must be explicitly declared to be NOT NULL in DB2.

* Missing information (unknown) is recorded as NULL null≠ blank,≠0.

* In DB2, any field can contain nulls unless the definition of that field explicitly specifies NOT NULL.

* Any column contains null value before specified unless it is defined as NOT NULL.

* Key attributes can be specified via the CREATE UNIQUE INDEX command.

ALTER TABLE base-table ADD column data-type;

ALTER TABLE S ADD DISCOUNT SMALLINT;

DROP TABLE base-table.

CREATE [UNIQUE] INDEX index
ON base-table (column [order] [, column [order]] ...)

[CLUSTER];

CREATE INDEX X ON T (P, Q DESC, R)CLUSTER;

CREATE UNIQUE INDEX XS ON S (S#);

CREATE UNIQUE INDEX XP ON P (P#);

CREATE UNIQUE INDEX XSP ON SP (S#, P#);

CREATE INDEX XSC ON S (CITY);

DROP INDEX index;

* Create index (order can be ASC -- ascending, or DESC - descending).

* SQL allows a table to have two or more tuples that are identical in all their attribute values.

* The unique option in CREATE INDEX specifies that no two records in the indexed base table will be allowed to take on the same value for the indexed field or field combination at the same time.

  DB2 will reject any attemp to introduce a duplicate value (via INSERT or UPDATE operation) into field S.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

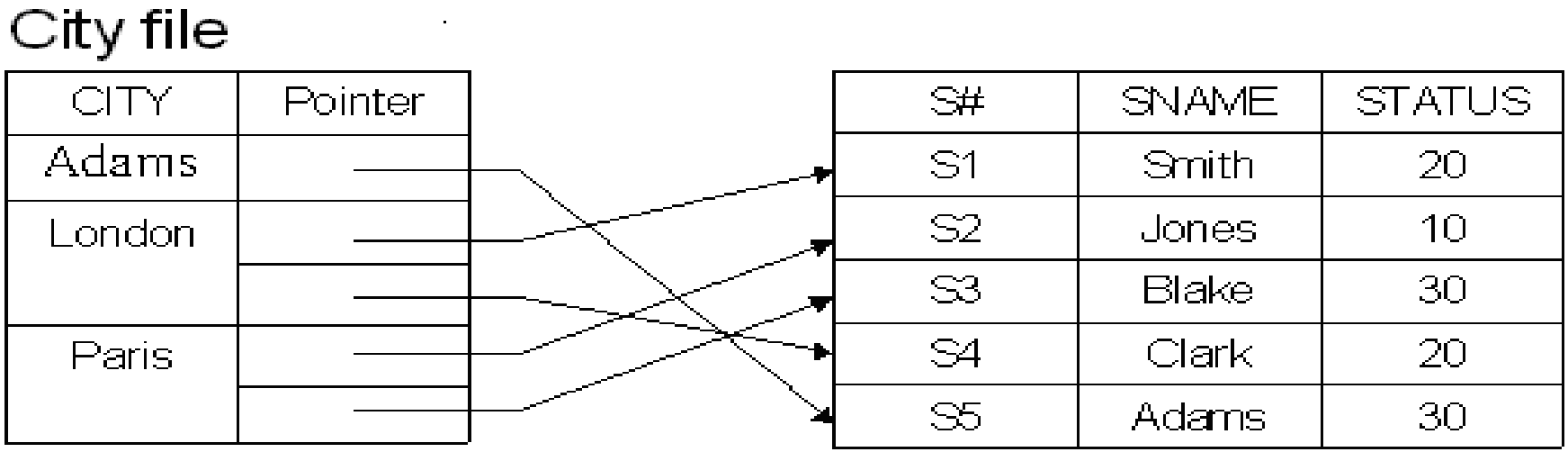

Fig. Indexing on CITY

SNAME index

SNAME	Pointer
Smith	S1
Jones	S2
Blake	S3
Clark	S4
Adams	S5

STATUS index

STATUS	Pointer
10	S2
20	S1, S4
30	S3, S5

CITY index

CITY	Pointer
Athens	S5
London	S1, S4
Paris	S2, S3

S file

S#
S1
S2
S3
S4
S5

Fig. Inverted Organization

DML（資料處理語言）

* Select, Update, Delete, Insert: return a set of data --> a set level language.

- Can operate at both the external and the conceptual level.

SELECT [DISTINCT] item(s)
FROM table(s)
[WHERE condition]
[GROUP BY field(s)]
{ Having condition }
{ ORDER BY field(s) };

* Distinct：eliminate duplicate rows.

* Where condition（＝, ＜＞, ＞, ＜, ＞, ≧, ≦, AND, OR, NOT）

* Examples：

```
SELECT S#, STATUS
 FROM S
WHERE CITY = 'Paris';
```

```
SELECT S.S#, S.STATUS
FROM S
WHERE CITY = 'Paris';
```

- Result：

S#	STATUS
S2	10
S3	30

* Simple Retrieval

```
SELECT P#                    SELECT DISTINCT P#
FROM SP;                     FROM SP;
```

* Retrieval of Computing Values

```
SELECT  P.P#, 'Weight in grams =', P.weight * 454
FROM P
```

* Simple Retrieval（SELECT *）

```
SELECT *
FROM S;
```

* Qualified Retrieval（WHERE）

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

Result：

S#
S3

* Retrieval With Ordering

```
SELECT S#, STATUS
FROM S
WHERE CITY = 'Paris'
ORDER BY STATUS DESC
```

(ORDER BY 3 DESC;)

Result :

S#	STATUS
S3	30
S2	10

* JOIN

- Join：a query in which data is retrieved from more than one table.
- Equijoin：Natural Join (with duplicate column eliminated)

- form the Cartesian product of the tables first.
- eliminate from this Cartesian product all those rows that do not satisfy the join condition.

R

A	B
1	b
2	b
3	c

S

B	C
a	4
b	5
b	6

```
SELECT R.A, R.B, S.C
FROM R,S
WHERE R.B=S.B ;
(R JOIN S)
```

1. R*S (CROSS PRODUCT)

A	R.B	S.B	C
1	b	a	4
1	b	b	5
1	b	b	6
2	b	a	4
2	b	b	5
2	b	b	6
3	c	a	4
3	c	b	5
3	c	b	6

2. Selection (R.B = S.B)

A	R.B	S.B	C
1	b	b	5
1	b	b	6
2	b	b	5
2	b	b	6

3. Projection A, B, C

A	B	C
1	b	5
1	b	6
2	b	5
2	b	6

Fig. The detail operation of JOIN

* Simple EQUIjoin

"Get all combinations of supplier and part information such that the supplier and part in question are located in the same city“

SELECT S.*, P.*

- FROM S, P

  WHERE S.CITY = P.CITY; (* Join condition *)

- Result：

S#	SNAME	STATUS	S.CITY	P#	PNAME	COLOR	WEIGHT	P.CITY
S1	Smith	20	London	P1	Nut	Red	12	London
S1	Smith	20	London	P4	Screw	Red	14	London
S1	Smith	20	London	P6	Cog	Red	19	London
S2	Jones	10	Paris	P2	Bolt	Green	17	Paris
S2	Jones	10	Paris	P5	Cam	Blue	12	Paris
S3	Blake	30	Paris	P2	Bolt	Green	17	Paris
S3	Blake	30	Paris	P5	Cam	Blue	12	Paris
S4	Clark	20	London	P1	Nut	Red	12	London
S4	Clark	20	London	P4	Screw	Red	14	London
S4	Clark	20	London	P6	Cog	Red	19	London

* Greater-Than Join（or less-than）

```
SELECT  S.*,  P.*
FROM  S, P
WHERE  S.CITY  >  P.CITY;
```

* Join Query With an Additional Condition

```
SELECT S.*, P.*
FROM S, P
WHERE  S.CITY = P.CITY  AND   S.STATUS <> 20
```

* Join of Three Tables

"Get all pairs of city names such that a supplier located in the first city supplies a part stored in the second city."

```
SELECT  DISTINCT  S.CITY,  P.CITY
FROM  S, SP, P
WHERE  S.S# = SP.S#  AND  SP.P# = P.P#;
```

Result：

S.CITY	P.CITY
London	London
London	Paris
London	Rome
Paris	London
Paris	Paris

* Join a Table With Itself；ALIAS

"Get all pairs of supplier numbers such that the two suppliers concerned are colocated."

```
SELECT  FIRST.S#,   SECOND.S#
FROM   S  FIRST,  S SECOND
WHERE   FIRST.CITY =  SECOND.CITY
      AND  FIRST.S#  <  SECOND.S#;
```

- Result：

S#	S#
S1	S4
S2	S3

* Aggregate Functions

SELECT COUNT(*)
FROM S;
Result： 5

SELECT COUNT (DISTINCT S#)
FROM SP;
Result：4

SELECT COUNT(*)
FROM SP
WHERE P# = 'P2';
Result： 4

* SUM, AVG, MAX, MIN: input -- one column; output – a single value.

SELECT SUM (QTY)
FROM SP
WHERE P# = 'P2';
Result： 1000

* Use of Group By

- Group By: logically rearranges the table into partition or groups, such that within any one group all rows have the same value for the Group By field.
- The Select clause is then applied to each group of the partitioned table.
- Each expression in the select clause must be single-value per group.

"For each part supplied, get the part number and the total shipment quantity for that part."

SELECT P#, SUM (QTY)

FROM SP

GROUP BY P#;

Result：

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

B* Use of Having

- Having：Having is to group (just like Where is to rows).
- Having is used to eliminate groups (just like Where is used to eliminate rows).
- Expression in a having clause must be single-valued per group.

"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

* Retrieval Using LIKE

```
SELECT   P.*
FROM   P
WHERE   P.PNAME LIKE 'C%';
(* column LIKE string-literal *)
```

Result :

P#	PNAME	COLOR	WRIGHT	CITY
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

* Query: For each project on which more than two employees work, retrieve the project number, project name, and the number of employees who work on the project.

```
SELECT      PNUMBER, PNAME, COUNT(*)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME
HAVING      COUNT(*) > 2;
```

PNAME	PNUMBER		ESSN	PNO	HOURS
ProductX	1		123456789	1	32.5
ProdictX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3	...	333445555	3	10.0
Computerization	10		333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	null
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

These groups are not selected by the having condition.

After applying the WHERE clause but before applying Having.

PNAME	PNUMBER		ESSN	PNO	HOURS
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	null
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

PNAME	COUNT(*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result

(PNUMBER not show)

After applying the Having clause condition.

* Retrieval Involving NULL

- The result of a comparison that involves NULL values is always false (> = < <>).

```
SELECT   S#
FROM   S                    (* not STATUS = NULL *)
WHERE    STATUS  IS NULL;
(* column-name  IS [NOT] NULL *)
```

* Retrieval Involving a Subquery

- IN: to test if the value in question exists in a set or not (boolean value result).
- Subquery: represent the set of values to be searched via an "IN condition".
- Subquery is evaluated first.

"Get supplier names for suppliers who supply part P2."

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

```
SELECT  SNAME
FROM  S
WHERE  S# IN
('S1', 'S2', 'S3', 'S4');
```

Result :

SNAME
Smith
Jones
Blake
Clark

```
SELECT  S.SNAME
FROM  S, SP
WHERE  S.S# = SP.S#
AND SP.P# = 'P2' ;
```

* Subquery With Multiple Levels of Nesting

"Get supplier names 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'));
```

（join？）

Result：

SNAME
Smith
Jones
Clark

* Subquery With Comparison Operator Other Than In

"Get supplier numbers for suppliers who are located in the same city as supplier S1.“

```
SELECT  S#
FROM  S
WHERE  CITY =
      ( SELECT  CITY          (* scalar comparison *)
      FROM  S
      WHERE S# = 'S1');
```

Result：

S#
S1
S4

* Aggregate Function In a Subquery

"Get supplier number 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

* Query Using EXISTS

- EXIST: to test if a set is empty or not (boolean value result).

"Get supplier names for suppliers who supply part P2."

```
SELECT  SNAME
FROM  S
WHERE EXISTS
        (SELECT  *
        FROM   SP
        WHERE  S# = S.S#  AND P# = 'P2');
```

* Query Using NOT EXISTS

"Get supplier names for suppliers who do not supply part P2."

```
SELECT SNAME
FROM S
WHERE NOT EXISTS
(SELECT *
     FROM SP
     WHERE S# = S.S#
           AND P# = 'P2');
```

```
SELECT  SNAME
FROM S
WHERE  S# NOT IN
     (SELECT  S#
      FROM SP
      WHERE  P# = 'P2');
```

Result :

SNAME
Adams

* Query Using NOT EXISTS

"Get supplier names for suppliers who supply all parts."

"Select supplier names for suppliers such that there does not exist a part that they do not supply."

```
SELECT  SNAME
FROM S
WHERE  NOT EXISTS
      (SELECT *
      FROM P
      WHERE  NOT EXISTS
            (SELECT *
            FROM SP
            WHERE S# = S.S#
                  AND   P# = P.P#));
```

Result :

SNAME
Smith

Query Involving UNION

"Get part number for parts that either weight more than 16 pounds or are supplied by supplier S2 (or both).“

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

Result：（P1, P2, P3, P6）

```
UPDATE  table
SET  field = scalar-expression [, field = scalar-expression] ...
[WHERE condition];
```

* Single-record UPDATE

```
UPDATE P
SET COLOR = 'Yellow',
WEIGHT = WEIGHT + 5,
CITY = NULL
WHERE P# = 'P2';
```

* Multiple-record UPDATE

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

```
DELETE
FROM table
WHERE condition;
```

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

```
INSERT
INTO table [(field [, field] ...)]
VALUES (literal [, literal ] ... );
```

```
INSERT
INTO SP (S#, P#, QTY)
VALUES ('S20', 'P20', 1000);
```

INSERT
INTO table [(field [, field] ...)]
subquery;

```
INSERT
INTO TEMP (P#, TOTQTY)
    SELECT P#, SUM(QTY)
    FROM SP
    GROUP BY P#;
```

* Nonprocedural language
  - Specify what (what they want), not how (without specifying a procedure for getting it).
  - Users do not navigate around the physical database to locate the desired data (it is done by the system).
* Query optimizer: its function is to choose an efficient access technique for implementing that statement.

View（景觀）

* A View is a derived virtual table (in SQL).
* External view in architecture of SQL = base table + view.
* A virtual table, a table that does not exist in its own right but looks to the user as if it did (while a base table is a real table).
* View's definition (in terms of other tables) is stored in the catalog.
* When this CREATE VIEW is executed, the subquery following the AS is not executed; instead, it is simply saved in the catalog.

* A view is like a window into the real table S.

  This window is dynamic: changes to S will be automatically and instantaneously visible through this window.

* Changes to view will automatically and instantaneously be applied to the real table S.

* The system handles DML (to view) such an operation by converting it into an equivalent operation on the underlying base table. --> new statement is complied, bound and executed.

* In AS statement, DB2 does not allow a view definition to include the UNION operator.

CREATE VIEW view [(column [, column] ...)]
AS subquery
[WITH CHECK OPTION];

* CREATE VIEW GOOD_SUPPLIERS
AS SELECT S#, STATUS, CITY
FROM S
WHERE STATUS > 15;

SELECT * FROM GOOD_SUPPLIERS WHERE CITY <> 'London';	SELECT S#, STATUS, CITY FROM S WHERE CITY <> 'London' AND STATUS > 15;

GOOD_SUPPLIERS

S#	STATUS	CITY
S1	20	London
S3	30	Paris
S4	20	London
S5	30	Athens

Result：

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

UPDATE GOOD_SUPPLIERS SET STATUS = STATUS + 10 WHERE CITY = 'Paris';	UPDATE S SET STATUS = STATUS + 10 WHERE CITY = 'Paris' AND STATUS > 15;

* CREATE VIEW PQ (P#, TOTQTY)
  AS SELECT P#, SUM (QTY)
  FROM SP
  GROUP BY P#;

* CREATE VIEW CITY_PAIRS (SCITY, PCITY)
  AS SELECT DINTINCT S.CITY, P.CITY
  FROM S, SP, P
  WHERE S.S# = SP.S#
  AND SP.P# = P.P#;

(a supplier located in city x supplies a part stored in city y.)

* CREATE VIEW REDPARTS (P#, PNAME, WT, CITY)
AS SELECT P#, PNAME, WEIGHT, CITY
FROM P
WHERE COLOR = 'Red';

* CREATE VIEW LONDON_REDPARTS
AS SELECT P#, WT
FROM REDPARTS
WHERE CITY = 'London';
(* define a view in terms of other views. *)

* CREATE VIEW GOOD_SUPPLIERS
AS SELECT S#, STATUS, CITY
FROM S
WHERE STATUS > 15
WITH CHECK OPTION; (* reject what is wrong *)

(* UPDATE AND INSERT operations against the view are to be checked to ensure that every UPDATEd or INSERTed row still satisfies the view-defining condition. *)

* DROP VIEW view；
  - Its definition is removed from the catalog.
  - When a table (base table or view) is dropped, all views defined on this table are automatically dropped too.
* Update a view？（* Not all views are updatable. *）

* A Column Subset View：primary key ??

CREATE VIEW S#_CITY
AS SELECT S#, CITY
FROM S;

---

1) INSERT (S6, Rome)
→(S6, NULL, NULL, Rome)
2) DELETE (S1, London)
→(S1, Smith, 20, London)
3) UPDATE → OK

CREATE VIEW STA_CITY
AS SELECT STATUS, CITY
FROM S;

INSERT (40, Rome)
→(Null, Null, 40, Rome) ??
DELETE (20, London) ??
→Which one ?
(20, London) to (20, Rome) ??

A Row Subset View: conflicting situation with the WHERE condition?

```
CREATE VIEW LONDON_SUPPLIERS
AS SELECT  S#, SNAME, STATUS, CITY
      FROM S
      WHERE CITY = 'London';
```

---

1) INSERT (S7, Peter, 30,' Paris') ?
   INSERT (S2, ..., 'London') ? (already exist !)
2) UPDATE (S1, .., 'London') to (S1, ..., 'Paris') ?
   (reject ? or disappear ?)
3) DELETE → OK.

* A Join View：trouble！！

```
CREATE VIEW COLOCATED (S#, SNAME,
STATUS,SCITY, P#, PNAME, COLOR, WEIGHT, PCITY)
AS  SELECT  S#, SNAME, STATUS, S.CITY,
                P#, PNAME, COLOR, WIEGHT, P.CITY
      FROM S, P
      WHERE S.CITY = P.CITY;
```

* A Statistical Summary View; trouble！！

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

* Physical data independence: user and user's programs are not dependent on the physical structure of the stored database.

* Logical data independence: users and user's programs are also independent of the logical structure (base table, conceptual view) of the database.

1. growth. The expansion of an existing base table to include a new field; or add a new base table.
2. restructuring. Ex. split a table vertically.
   (S#, sname, city) and (S#, status)

```
create view S (S#, sname, status, city)
as select SX.S#, SX.sname, SY.status, SX.city
from SX, SY
where SX.S# = SY.S#;
```

Advantages：

1. provide a certain amount of logical data independence in the face of restructuring.
2. allow the same data to be seen by different users in different ways (at the same time).
3. the user's perception is simplified. (focus on what they want)
4. automatic security is provided for hidden data.

Homework

Translate all queries into SQL format for the following database.

EMP (emp#, name, dept#, job, manager#, salary, age)

DEP (dept#, dname, location)

CANDIDATE (emp#, name, dept#, salary)

1. List employee name and his (her) department location if this employee's salary is greater than 15000.

2. List employee number, name and salary of employees in department 40, in order of employee number.

3. Find the average salary of clerks (a kind of job) whose age is older than 30.

4. How many different jobs are held by employees in department 40 ?

5. List all the jobs and the average salary of each.

6. List the names of all employees and the locations of their departments.

7. For each employee whose salary exceeds his or her manager's salary, list the employee's name and the manager's name.

8. For all departments in Columbus with average salary < 20000, list the department number and average salary ordered by average salary in the descending order.

9. Delete the department having no employees from the DEP table.

10. Update the EMP table by giving a ten percent raise (salary) to all those whose employee number appears in the Candidate table.