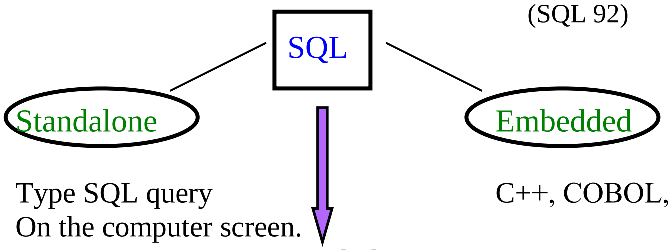
Please see below

SQL → Structured Query Language

New Version SQL3
→ Object oriented

SQUARE → SEQUEL → SQL → SQL2 → SQL3
(SQL 92)



Type SQL query
On the computer screen.

C++, COBOL, Java…

Includes
Statement like Select, Insert, Delete, Update, Create table (data definition), etc.

SELECT – retrieves data from the database.

SELECT the result COLUMN
FROM the source TABLE
WHERE condition <, =…

STUDENT:

SS#	NAME	MAJOR
123	TOM	CIS
200	AMY	CIS
210	BILL	FIN
220	GARY	ACC

To avoid getting duplicates use the qualifier ***DISTINCT*** after **SELECT.**

EG: SELECT DISTINCT MAJOR
FROM STUDENT

RESULT: CIS, FIN, ACC

Query #1*:* What are the names of CIS majors?

Result:

NAME
TOM
AMY

SELECT NAME
FROM STUDENT
WHERE MAJOR = 'CIS'

Conditions are in the form of:
Attribute **comparison** constant or attribute.

SELECT examines each tuple.

=, <>
>, >=
<, <=
Connectors: AND, OR, NOT.

SUPPLIER:

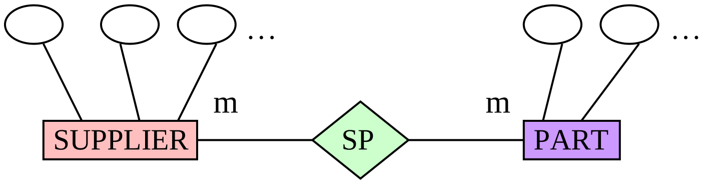
S#	SNAME	CITY	STATUS
S1	IBM	NY	10
S2	KLM	AMS	20
S3	ATT	NY	10
S4	GTE	LA	30
S5	MCI	NY	10

SP:

S#	P#	QTY
S1	P1	30
S1	P2	40
S1	P3	10
S1	P4	10
S2	P1	30
S2	P2	40
S3	P1	40
S3	P3	50
S4	P4	40

PART:

P#	PNAME	COLOR	WEIGHT
P1	PC/XT	BLACK	10
P2	PS/2	RED	20
P3	BOLT	BLACK	5
P4	PIN	BLUE	1



- For the many to many (m: m) relationships define a separate table.

In the case of one to many (1: m) combine it with the entity table at the many end of relationship to reduce data redundancy.

Query #1: Get the P# for the parts that are supplied.

SELECT P#
FROM SP

RESULT:

P#
P1
P2
P3
P4
P1
P2
P1
P3
P4

The result **looks like a relation but is not** because, *in a relation*, duplicates are not allowed

Multi-set means there are duplicates rows

SQL has an anomaly since it does not eliminate duplicates.
Use **DISTINCT** to eliminate the duplicates.

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

RESULT:

P#	
P1	~~P2~~
P2	~~P1~~
P3	~~P3~~
P4	~~P4~~

Query #2: Get all the attributes of suppliers.

```
SELECT S#, SNAME, CITY, STATUS
FROM   SUPPLIER
```

Or (*) Means all the attributes.

```
SELECT  *
FROM   SUPPLIER
```

Query #3: Get suppliers in NY whose statuses is less than 20.

```
SELECT  *
FROM   SUPPLIER
WHERE   CITY =‘NY’ AND STATUS < 20
```

RESULT:

S#	SNAME	CITY	STATUS
S1	IBM	NY	10
S3	ATT	NY	10
S5	MCI	NY	10

Query #4: Same as #3, result is in ascending order with respect to supplier name.

```
SELECT  *
FROM SUPPLIER
WHERE CITY= ‘NY’ AND STATUS < 20
ORDER BY SNAME ASCD
```

RESULT:

S#	SNAME	CITY	STATUS
S3	ATT	NY	10
S1	IBM	NY	10
S5	MCI	NY	10

Although in relations the order is not important, for the convenience of the user SQL includes ORDER BY clause.

Query #5: Get the P# and QTY for the parts supplied by the suppliers located in NY.

SELECT P#, QTY
FROM SUPPLIER, SP

Cartesian product
Use as many relations as you need separated by commas. The order does not matter.

WHERE CITY= 'NY' **AND**
SUPPLIER.S# = SP.S#

corresponds to *Join operation*
SUPPLIER.S# = SP. SP#

$$\underbrace{\text{CITY= 'NY'}}\ \underbrace{\textbf{AND}\text{ SUPPLIER.S\# = SP.S\#}}$$

NY = NY **AND** S1 = S1 →T
NY = NY **AND** S1 = S1 →T
:
:
NY = NY **AND** SI = S2 →F

RESULT:

P#	QTY
P1	30
P2	40
P3	10
P4	10
P1	40
P3	50

Each tuple of SUPPLIER and SP are examined for all the possible combinations.
The corresponding relational algebra expression:

$$\overbrace{\pi_{\text{P\#, QTY}}}^{\text{A3}}((\overbrace{\sigma_{\text{CITY='NY'}}\text{ (S}}^{\text{A1}}\overbrace{\text{UPPLEIR))} \bowtie_{\text{SUPPLIER.S\# = SP.SP\#}} \text{SP)))}}^{\text{A2}}$$

A1. CITY= 'NY' → $(\sigma_{\text{CITY = 'NY'}}\text{(SUPPLIER)})$

A2. SUPPLIER.S# = SP.S# → $\text{(A1} \bowtie_{\text{A1.S\# = SP.SP\#}} \text{SP))}$

A3. P#, QTY → $\pi_{\text{P\#, QTY}}\text{ (A2)}$

Query #6: What are the names of parts supplied by the suppliers located in NY?

SELECT PNAME
FROM SUPPLIER, PART, SP
WHERE CITY = 'NY' **AND**
SUPPLIER.S# = SP. S# **AND** → join SUPLIER and SP.
SP. P# = PART. P# → join SP and PART.

To shorten **SUPPLIER.S# = SP. S#**; you may create a new name (ALIAS) for the relation names PART (P) and SUPPLIER (S).

```
SELECT PNAME
FROM SUPPLIER S, PART P, SP
WHERE CITY = 'NY' AND
         S. S#  = SP. S# AND
        SP. S#  = P. P#
```

Query #7: Get the pairs of S#'s such that the 2 suppliers are located in the same city.

Desired result

S#	S#	
S1	S3	
S1	S5	
S3	S5	
~~S3~~	~~S1~~	Redundant

```
SELECT S#, S#
FROM    SUPPLIER, SUPPLIER
WHERE CITY = 'NY' AND
                S# <> S#
```

Ambiguous

BY USING **ALIASES** WE SOLVE THE AMBIGUITY PROBLEM!

```
SELECT X. S#, Y. S#

FROM SUPPLIER X, SUPPLIER Y

WHERE X.CITY = Y. CITY   AND
             X .S#  < Y.S#
```

SUPPLIER X

S1	NY
S2	AMS
S3	NY
S4	LA
S5	NY

JOIN
CITY = CITY

SUPPLIER Y

S1	NY
S2	AMS
S3	NY
S4	LA
S5	NY

WHERE X.CITY = Y.CITY **AND**

⌘X.S# <> Y.S# Eliminates tuples like < S1 S1 >

✪X. S# < Y.S# Eliminates tuples like < S3 S1>

RESULT:

X. S#	**Y. S#**	
~~S1~~	~~SI~~	⌘
S1	S3	
S1	S5	
~~S2~~	~~S2~~	⌘
~~S3~~	~~S1~~	✪
~~S3~~	~~S3~~	⌘
S3	S5	
~~S4~~	~~S4~~	⌘
~~S5~~	~~S1~~	✪
~~S5~~	~~S3~~	✪
~~S5~~	~~S5~~	

Final result is:

X. S#	**Y. S#**
S1	S3
S1	S5
S3	S5

Query #8: What are the names of suppliers who supply parts in a QTY greater than 30?

```
SELECT SNAME
FROM    SUPPLIER, SP
WHERE QTY>30 AND
            SUPPLIER. S# = SP. S#
```

RESULT: IBM
KLM
ATT
GTE

Another SQL expression for this query is:
In plain English this means:

(S1, S2, S3, S3, S4)

***SNAMES** of*
***SUPPLIER** who*
are among the

```
SELLECT SNAME
FROM       SUPPLIER
WHERE S# IN  (SELECT S#
              FROM SP
              WHERE QTY>30)
```

***S#** of **SUPPLIER** who*
*supply in a **QTY** greater than 30*

Note: It is logical to process the inner most select statement first.

Query #9: What are the names of suppliers who supply parts **only** in a QTY greater than 20?

```
SELECT SNAME
FROM SUPPLIER, SP
WHERE QTY > 20   AND
             SUPPLIER. S# = SP. S#
```

(#)
This SQL statement is incorrect
because:

RESULT:
<u>SNAME</u>

IBM
KLM
ATT
GTE

IBM
supplies quantities greater than and less than 20.

SNAME of SUPPLIER
where

```
SELECT SNAME
FROM   SUPPLIER

WHERE S# IN   (SELECT S#
               FROM   SP
               WHERE  QTY > 20 AND

               S# NOT IN  (SELECT S#
                           FROM SP
                           WHERE QTY <= 20)
```

(IBM, KLM, ATT, GTE)
(S1, S1, S2, S2, S3, S3, S4)

Incorrect:

It gives the same SNAMES (#),
BUT if we add another condition:

SUPPLIER supply in a QTY > 20

(**S1, S1**)

And that supplier is not
among the suppliers *who supply in*
a quantity < = 20. We obtain the
correct Result

SNAME: **KLM, ATT, and GTE**

Query #10: What are the names of suppliers who **do not** supply any parts?

*DON'T use **join*** **operation**

```
SELECT SNAME
FROM SUPPLIER, SP
WHERE SUPPLIER.S# <> SP.S#
```

RESULT:

SNAME
IBM
IBM
KLM
ATT
ATT
MCI…

This SQL statement corresponds to the following relational algebra expression:

$$\pi_{\text{SNAME}} (\text{SUPPLIER} \bowtie_{\text{SUPPLIER.S\#} \neq \text{SP.S\#}} \text{SP.S\#})$$

With this expression you still get the same answer as with the SQL query, it is also incorrect.
The correct expression is:

$$\pi_{\text{SNAME}} (\text{SUPPLIER} \bowtie \underbrace{(\pi_{\text{S\#}} (\text{SUPPLIER})}_{} - \underbrace{(\pi_{\text{S\#}} (\text{SP}))))}_{}$$

S# of suppliers that
exist in the SUPPLIER
(S1, S2, S3, S4, S5)

S# of suppliers who supply
at least 1 part.
(S1, S2, S3, S4)

RESULT: SNAME
MCI

After doing the subtraction of the two relational algebra expressions above we obtain the desired result.

The correct expression:
SNAME of suppliers who are NOT IN (among) the → *S# of suppliers supplying some parts*

```
SELECT SNAME
 FROM  SUPPLIER
```

(S1, S1, S1, S1, S2, S2, S3, S3, S4)

```
WHERE S# NOT IN  (SELECT S#
                    FROM SP)
```

Another SQL Expression For the query #9

```
SELECT SNAME
FROM SUPPLIER
WHERE S# NOT IN  (SELECT S#
                   FROM SP
                  WHERE QTY < = 20) AND
                      S# IN (SELECT S#
                        FROM SP)
```

Note that in this expression there is double negation of the conditions:
i.e., S# NOT IN,
NOT (QTY > 20) → QTY <= 20

Condition:

1. Attribute **comparison** attribute or constant

=, <>, >, >=, <, <=, Like, between

2. Attribute [NOT] IN set

- (Constant, constant, …)
- (Select statement) → Returns a relation

Optional

Query #11: What are the names of the suppliers located in AMS **or** in LA?

SELECT SNAME
FROM SUPPLIER
WHERE CITY = 'AMS' **OR** CITY = 'LA'

or

```
SELECT SNAME
FROM SUPPLIER
WHERE CITY IN (‘AMS’, ‘LA’)
```

***Query #12**:* What are the names of suppliers who supply parts P3 or P4?
Alternative I:

```
SELECT SNAME
FROM    SUPPLIER, SP
WHERE SUPPLIER.S# = SP.S# AND P# IN  (‘P3’, ‘P4’)
```

Alternative II:

(S1, S1, S3, S4)

SNAMES such that their S# are in S# that supply P3 or P4

```
SELECT SNAME
FROM SUPPLIER
WHERE S# IN  (SELECT  S#
              FROM SP
              WHERE P# = ‘P3’ OR P# = ‘P4’)
```

RESULT:

SNAME
IBM
ATT
GTE

Query #13: What are the names of suppliers who **do not** supply any parts? –This is a repetition query #10

CONDITIONS in WHERE clause

1. EXISTS (SET)
NOT EXISTS (SET)

2. Attribute **Comp**.Attribute **or** constant
Comp.
=, <>, …
LIKE→for strings
BETWEEN→ **f**or ranges

3. [NOT] IN set

This **set is not empty** there is at least one element therefore, it EXISTS.

By EXISTS we mean whether the set is empty or not.

(Empty) or
(Not empty)

Ex: EXISTS (5, 10, 20) → T
EXISTS (5) → T
EXISTS () → F
NOT EXISTS (5, 10, 20) → F
NOT EXISTS (5) → F
NOT EXISTS () → T

1. **EXISTS** and **NOT EXISTS** are operators whose condition is either true of false depending on the presence or absence of rows in the set following EXISTS OR NOT EXISTS.

2. **LIKE:** a SQL keyword to test partial values in strings (subatrings).

***** = Matches to any number of characters. Ex: CITY LIKE 'N*****' → NY, New York, NJ, Nx.

? = Matches exactly to one character. Ex. CYTY LIKE 'N**?**' →NY, NJ, NH, etc.

# = Matches to one digit. Ex. ZIP LIKE '**# # # # #**'

CITY **LIKE** '[A, B-E] * '→this example matches city with string that starts with A or B through E, followed by any number of characters, e.g., Albany, Edison, but not City.

LIKE → CHAR fixed (3) N Y _ Blank space
LIKE → VARCHAR (3) N Y
A M S

BETWEEN → **RANGE** → STATUS **BETWEEN** 10 **AND** 20 is the same as STATUS >= 10 **AND** STATUS <= 20

Relational Schema

SUPPLIER:

S#	SNAME	CITY	STATUS
S1	IBM	NY	10
S2	KLM	AMS	20
S3	ATT	NY	10
S4	GTE	LA	30
S5	MCI	NY	20

SP:

S#	P#	QTY
S1	P1	30
S1	P2	40
S1	P3	10
S1	P4	20
S2	P1	20
S2	P2	30
S3	P1	40
S3	P3	50
S4	P1	40

PART:

P#	PNAME	COLOR	WEIGHT
P1	PC/XT	BLACK	10
P2	PS/2	RED	20
P3	BD/LT	BLACK	5
P4	PIN	BLUE	1

Query 14. What are the P# of parts supplied by suppliers located in NY?

Alternative 1:

```
SELECT      P#
FROM        SP
WHERE       S# IN (SELECT     S#
                   FROM       SUPPLIER
                   WHERE      CITY = 'NY')
```

Alternative 2:

```
SELECT      P#
FROM        SP, SUPPLIER
WHERE       CITY = 'NY' AND
            SP. S# = SUPPLIER. S#
```

Alternative 3:

```
SELECT      P#
FROM        SP
WHERE       EXISTS (SELECT   *
                   FROM SUPPLIER
                   WHERE CITY = 'NY' AND
                      SUPPLIER. S# = SP.S#)
```

this is an example of **correlated query**

Correlated variable →

S# P# QTY		S#			RESULT:
<S1, **P1**, 30>	→	(S1, IBM, NY, 10)	→	**T**	**P1**
<S1, **P2**, 40>	→	(S1, IBM, NY, 10)	→	**T**	**P2**
<S1, **P3**, 10>	→	(S1, IBM, NY, 10)	→	**T**	**P3**
<S1, **P4**, 20>	→	(S1, IBM, NY, 10)	→	**T**	**P4**
<S3, **P1**, 40>	→	(S3, ATT, NY, 10)	→	**T**	**P1**
<S3, **P3**, 50>	→	(S3, ATT, NY, 10)	→	**T**	**P3**

All the other tuples return an empty set and the EXISTS condition evaluates to False. For a tuple in SP all the tuples in SUPPLIER are considered.

Query #15: What are the names of suppliers who supply parts in a QTY more than 30?

```
SELECT SNAME
FROM   SUPPLIER
WHERE EXISTS (SELECT *                          RESULT: IBM, ATT, GTE
                   FROM   SP
                   WHERE QTY > 30 AND
                             SP. S# = SUPPLIER. S#) <——— what happens we omit it?

                             SP. S# = S1
<S1 IBM NY 10>               (S1 P1 30)        F                  }
                             (S1 P2 40)        T   (S1 P2 40)     }  T
                             (S1 P3 10)        F                  }
                             (S1 P4 20)        F                  }
               The remaining tuples of SP also make this condition false
               and are not included for the sake of brevity since their S# are
               different then S1.
..........................................................................
                             SP. S# = S2
<S2 KLM AMS 20>              (S2 P1 20)        F       Ø            }  F
                             (S2 P2 30)        F                    }
..........................................................................
                             SP. S# = S3
<S3 ATT NY 10>               (S3 P1 40)        T   (S3 P1 40)  }
                             (S3 P5 50)        T   (S3 P5 50)  } T
..........................................................................
                             SP. S# = S4
<S4 GTE LA 10>               (S4 P1 40)        T   (S4 P1 40)  }  T
..........................................................................
                             SP. S# = S5
<S5 MCI NY 10>                                       Ø         }  F
```

Query #16: Get suppliers who do not supply any parts

```
SELECT SNAME
FROM   SUPPLIER
WHERE NOT EXIST (SELECT *
                      FROM SP
                      WHERE SP.S# = SUPPLIER. S#)

                           SP.S# = S1
                         { (S1 P1 30)
<S1 IBM NY 10>           { (S1 P2 40)
                         { (S1 P3 10)                FALSE
                         { (S1 P4 20)
..........................................................................
                           SP. S# = S2
<S2 KLM AMS 20>         {  (S2 P1 20)
```

	(S2 P2 30)	FALSE
	SP. S# = S3	
<S3 ATT NY 10>	(S3 P1 40)	
	(S3 P3 50)	FALSE
	SP. S# = S4	
<S4 GTE LA 10>	(S4 P1 40)	FALSE
	SP. S# = S5	
<S5 MCI NY 10>	Ø EMPTY SET	TRUE

RESULT: SNAME

MCI

Query #17: What are the names of suppliers who supply all the parts?

```
SELECT SNAME
FROM    SUPPLIER
WHERE NOT EXIST (SELECT *
                  FROM PART
                  WHERE NOT EXIST  (SELECT *
                                     FROM SP
                                     WHERE SP.P# = PART.P# AND
                                                SP.S# = SUPPLIER.S#) )
```

SNAME such that ← there does not exist a part

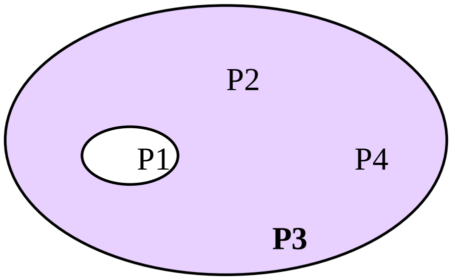
that is not being supplied ← by that name (SUPPLIER)

Supplier S1 supplies all the parts

There does not exist a part that is not supplied by S1

Both sentences mean the SAME thing.

I will use this figure In class



Aggregate functions

They are a set of five SQL built-in functions: COUNT, SUM, AVG, MAX, and MIN.

COUNT: computes the number of rows in a table.

SUM: totals numeric columns.

AVG, **MAX**, and **MIN:** also operate on numeric columns.

AVG: computes the average value of a column.

MAX and **MIN:** obtain the maximum and minimum values of a column in a table.

SUPPLIER: Relational Schema

S#	SNAME	CITY	STATUS
S1	IBM	NY	10
S2	KLM	AMS	20
S3	ATT	NY	10
S4	GTE	LA	30
S5	MCI	NY	20

SP:

S#	P#	QTY
S1	P1	30
S1	P2	40
S1	P3	10
S1	P4	20
S2	P1	20
S2	P2	30
S3	P1	40
S3	P3	50
S4	P1	40

PART:

P#	PNAME	COLOR	WEIGHT
P1	PC/XT	BLACK	10
P2	PS/2	RED	20
P3	BOLT	BLACK	5
P4	PIN	BLUE	1

Basic format

SELECT Aggregate function (Attribute)
FROM Source
WHERE Condition.

Query #1: What is the MAXIMUN quantity supplied by S1?

SELECT MAX(QTY)
FROM SP
WHERE S# = 'S1'

Result**:** **MAX**(QTY)
40

Query #2: What is the AVERAGE quantity supplied by S1?

SELECT AVG(QTY)
FROM SP
WHERE S# = 'S1'

RESULT: 25

Query #3: COUNT the number of parts supplied by S1?

SELECT COUNT(QTY) → COUNT(*)

```
FROM SP
WHERE S# = ‘S1’
```

RESULT: 4

Query #5: How many suppliers and part combinations are there?

```
SELECT COUNT (*)
FROM SP
```

RESULT: 9

To be more accurate and avoid repetition: - How many parts are supplied?

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

RESULT: 4

Aggregate Functions and Grouping (GROUP BY)

Aggregate functions can be applied to groups of rows within a table that have the same value in the specified columns.

Query #1: For each supplier get the total QTY

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

Result:

S#	SUM (QTY)
S1	100
S2	50
S3	90
S4	40

SP:

S#	P#	QTY
S1	P1	30
S1	P2	40
S1	P3	10
S1	P4	20
S2	P1	20
S2	P2	30
S3	P1	40
S3	P3	50
S4	P1	40

Query #2: For each supplier gets the total QTY, if the total QTY is more than 50

```
SELECT S#, SUM (QTY)
FROM SP
GROUP BY  S#
HAVING  SUM (QTY) >50
```

S#	SUM (QTY)
S1	100
~~S2~~	~~50~~
S3	90
~~S4~~	~~40~~

HAVING
Eliminates
the S# with SUM(QTY) >50

SQL **HAVING** clause, works on groups (partitions) unlike WHERE that works on rows.

Query #3: For each part get the total quantity if the total quantity is more than 50.

```
SELECT P#, SUM (QTY)
FROM SP
GROUP BY P#
HAVING SUM (QTY)>50
```

P#	SUM (QTY)
P1	130
P2	70
P3	60
~~P4~~	~~20~~ → Doesn't qualify

SP:

S#	P#	QTY
S1	P1	30
S1	P2	40
S1	P3	10
S1	P4	20
S2	P1	20
S2	P2	30
S3	P1	40
S3	P3	50
S4	P1	40

Query #4: For each supplier get the total quantity for the parts that suppliers supply in a quantity greater than 20?

```
SELECT S#, SUM (QTY)
FROM SP
WHERE QTY >20
GROUP BY S#
```

S#	SUM (QTY)
S1	70
S2	30
S3	90
S4	40

Query#5: For each supplier(S#) get the total quantity for the parts that suppliers supply in a quantity greater than 20. Do this query only if the average quantity QTY for supplier is more than 40.

```
SELECT S#, SUM (QTY)
FROM SP
WHERE QTY >20
GROUP BY  S#
```

S#	SUM (QTY)
S3	90

HAVING AVG (QTY) >40

Query#6: For each supplier (SNAME), get the total quantity for the parts that suppliers supply in a quantity greater than 20. Do this query only if the average quantity QTY for supplier is more than 40. Now instead of S# use SNAME in your query.

SELECT SNAME, SUM (QTY)
FROM SUPPLIER, SP
WHERE SUPPLIER. S# = SP.S# **AND**
QTY >20
GROUP BY SNAME
HAVING AVG (QTY) >40

What is the relational algebra expression for this query? Mimic how SQL computes the result of this query. Show the intermediate results if there is any.

Query #7: Get the SNAME of the supplier whose status is larger that the average status.

SELECT SNAME, STATUS
FROM SUPPLIER
WHERE STATUS > (**SELECT** AVG (STATUS)
FROM SUPLIER))

RESULT:

SNAME	STATUS
KLM	20
GTE	30
MCI	20

>, >=
<, <=
=, < > ONLY if we know that the inner select statement is going to return one value.

UPDATE: UPDATE relation name
SET attributes = value
WHERE condition

Query #8: Increase the QTY supplied by S1 by 50%

UPDATE SP
SET QTY = 1.50 * QTY → (QTY + O.5 * QTY)
WHERE S# = 'S1' (if no WHERE clause, all the values in SP will be increased not just S1)
Query #9: Supplier S5 supplies Part P1 in a quantity of 50.

INSERT INTO SP

VALUES (‘S5’, ‘P1’, 50)

SP:

S#	P#	QTY
S1	P1	30
S1	P2	40
S1	P3	10
S1	P4	20
S2	P1	20
S2	P2	30
S3	P1	40
S3	P3	50
S4	P1	40
S5	**P1**	**50**

Query #10: S1 no longer supplies P1

DELETE SP
WHERE S# = ‘S1’ **AND** P# = ‘P1’

SP:

S#	P#	QTY
S1	P2	40
S1	P3	10
S1	P4	20
S2	P1	20
S2	P2	30
S3	P1	40
S3	P3	50
S4	P1	40
S5	P1	50

Data definitions statements in SQL

CREATE A TABLE SP (
S# Char (3) NOT NULL,
P# Char (3) NOT NULL,
QTY INTEGER,
Primary key (S#, P#)
)

Note: we must enforce referential integrity.

Other Data definitions Statements in SQL include:

ALTER TABLE
DROP TABLE…