Master e-business and ICT Management

Database

SQL

Tania Cerquitelli

A.A. 2007 / 2008

Supplier and part 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

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

Supplier and part database

- Relational model
  - part base table
  - supplier base table
  - supply base table, that puts a relation between parts and suppliers which supply parts
- Primary key (record identifier)
  - part: P#
  - supplier: S#
  - supply: (S#,P#)

SQL language

- It is a language to define the relational database structure and to access and update data
- It is suitable for
  - *interactive operations*
  - *embedded instructions*
    - a host language contains SQL instructions. These instructions differ from host language instructions by means of syntactic artifices

SQL language

- SQL is a level set language
  - operations and results are set of data
- SQL expresses queries in declarative way
  - the abstraction level is higher than programming languages
  - queries specify the properties of the result, not the way to obtain it
  - queries are translated by the query optimizer into the procedural language internal to the DBMS
    - the programmer should focus on readability, not on efficiency

SQL language

- The name is an acronym for Structured Query Language
- It is a language to manage relational database
  - DDL (Data Definition Language) is the language which supports the definition or declaration of database objects (e.g., tables, indexes)
  - DML (Data Manipulation Language) is the language which supports the manipulation or processing of data

Data Definition Language

- CREATE TABLE: base table definition
- ALTER TABLE: alteration of a base table structure
- DROP TABLE: base table deletion
- CREATE INDEX: index creation
- DROP INDEX: index deletion

DBMG 7

CREATE TABLE

CREATE TABLE <base-table-name>
(column-definition [,column-definition]);
column-definition::=column-name type-name [NOT NULL]

Example. Create the supplier base table.

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

```
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#)
);
```

DBMG 8

Definition of supplier and part database

```
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#) REFERENCES S,
                  FOREIGN KEY (P#) REFERENCES P);
```

DBMG 9

ALTER TABLE

- The following “alterations” are supported
  - A new column can be added
  - A new default can be defined for an existing column (replacing the previous one, if any)
  - An existing column default can be deleted
  - An existing column can be deleted
  - A new integrity constraint can be specified
  - An existing integrity constraint can be deleted

DBMG 10

ALTER TABLE

ALTER TABLE <base-table-name>
ADD column-name data-type;

Example. Add the column DISCOUNT to supplier base table.

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

```
ALTER TABLE S
 ADD DISCOUNT SMALLINT;
```

DBMG 11

DROP TABLE

DROP TABLE <base-table-name>;

- indexes and views, defined on base-table-name, are also deleted

Example. Delete the supplier base table.

```
DROP TABLE S;
```

DBMG 12

INDEXES - Creation

CREATE [UNIQUE] INDEX <*index-name*> ON <*base-table-name*>(column-name [,column-name]);

- UNIQUE option specified than only one record can assume a given value

Examples.
CREATE UNIQUE INDEX XS ON S (S#);
CREATE UNIQUE INDEX XP ON P (P#);
CREATE UNIQUE INDEX XSP ON SP (S#,P#);
The violation of the unique option prevents the operation

CREATE INDEX XSC ON S (CITY);
Since many records can do reference to the same city, UNIQUE option is not specified

DBMG 13

INDEXES – Deleting

DROP INDEX index-name;

Example. Delete XSC index.

DROP INDEX XSC;

DBMG 14

Data Manipulation Language

- SELECT – retrieval data (querying database)
- INSERT – inserting new data
- UPDATE – changing existing data
- DELETE – deleting exiting data

DBMG 15

Query

SELECT [DISTINCT] column(s)
FROM table(s)
[WHERE condition]
[GROUP BY column(s)
[HAVING condition]]
[ORDER BY column(s)];

DBMG 16

Data manipulation

Example. Find the code and status relating to suppliers in Paris.

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

S base table

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

Result

S#	STATUS
S2	10
S3	30

DBMG 17

SELECT with DISTINCT

DISTINCT – Remove duplicates

Example. Find distinct supplied parts.

SELECT DISTINCT P#
FROM SP;

SP base table

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

Result

P#
P1
P2
P3
P4
P5
P6

DBMG 18

Extraction of all information

Example. Find all the information relating to parts.

```
SELECT *
FROM P;
```

```
SELECT P.P#, P.NAME, P.COLOR, P.WEIGHT, P.CITY
FROM P;
```

P base table

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

Result

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

DBMG 19

Ordering

- The ORDER BY clause, at the end of the query, orders the rows of the result

ORDER BY Ordering-Attribute [asc|desc]
{, Ordering-Attribute [asc|desc]}

Example. Extract the content of the part base table in descending order of name and color.

```
SELECT *
FROM P
ORDER BY PNAME DESC,
         COLOR DESC;
```

Result

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

DBMG 20

ORDER BY

Example. Get p# of all parts in descending order of p#.

```
SELECT P#
FROM P
ORDER BY P# DESC;
```

DESCENDING

Without DESC → Implicit order is ASCENDING

P base table

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

Result

P#
P6
P5
P4
P3
P2
P1

DBMG 21

Search with Ordering (1)

Example. Find all the information relating to parts. The result needs to be ordered in descending order of weight and ascending order of name.

```
SELECT P#, PNAME, COLOR, WEIGHT, CITY
FROM P
ORDER BY WEIGHT DESC, PNAME;
```

P base table

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

Result

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

DBMG 22

Search with Ordering (2)

Example. For each part find p# and weight expressed in gram. Order the result based on weight expressed in gram.

```
SELECT P#, WEIGHT*454 AS PESO
FROM P
ORDER BY PESO;
```

P base table

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

Result

P#	PESO
P1	5448
P5	5448
P4	6356
P2	7718
P3	7718
P6	8626

DBMG 23

Qualified search (Predicate conjunction)

Example. Find s# relating to suppliers in Paris with status greater than 20.

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

S base table

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

Result

S#
S3

DBMG 24

Queries (1)

1. Find s# and status relating to suppliers in Paris. Order the result with respect to status.

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

2. Find s# and status relating to suppliers in Paris or in London.

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

DBMG 25

Queries (2)

3. Find s# and status relating to suppliers without buildings in Paris.

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

DBMG 26

Query with JOIN

- This type of query searches a set of data in two or more base tables
- It performs a cartesian product of a set of base tables listed
- The result table is composed by all possible rows r, where r is obtained linking together one row of the first table, one for the second, ... , one of the last table
- All rows, which do not satisfy conditions expressed in WHERE, are removed from the result table

Example. Find the name of suppliers which supply the P2 part

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

↑ Join condition

DBMG 27

Query with JOIN

Example. Find names of suppliers which supply at least a red part

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

If in the FROM clause there are N base tables → At least N-1 Join Conditions

DISTINCT is necessary to remove duplicate

DBMG 28

Query with JOIN

Example. Find the pairs of s# relating to suppliers with building in the same city.

```
SELECT SX.S#, SY.S#
FROM S AS SX, S AS SY
WHERE  SX.CITY=SY.CITY AND SX.S#<>SY.S# ;
```

Result (a)

SX. S#	SY.S#
S1	S1
S1	S4
S2	S2
S2	S3
S3	S2
S3	S3
S4	S1

Result (b)

SX. S#	SY.S#
~~S1~~	~~S1~~
S1	S4
~~S2~~	~~S2~~
~~S2~~	~~S3~~
S3	S2
~~S3~~	~~S3~~
~~S4~~	~~S1~~

DBMG 29

Aggregate function

- The result of an aggregate query depends on the consideration of sets of rows
- SQL-2 offers five aggregate operators
  - COUNT – counts elements of a column
  - SUM – sums values of a column
  - AVG – value average of a column
  - MAX – maximum value in a column
  - MIN – minimum value in a column

DBMG 30

Operator COUNT

- COUNT returns the number of rows or distinct values

  COUNT (<*| distinct | all Attribute-List>)

- Function argument may be preceded by DISTINCT
- COUNT(*) counts the number of base table rows

Operator COUNT

1. Get the number of rows of S base table.

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

2. Get the number of suppliers which provide at least one supply.

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

3. Get the number of suppliers which supply the P2 part.

```
SELECT COUNT(*)
  FROM SP
  WHERE P#='P2';
```

4. Get the total number of supplied parts relating to P2 part.

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


SUM, AVERAGE, MAXIMUN, MINIMUM

< SUM | MAX | MIN | AVG >
([distinct | all] Attribute-Expression>)

- SUM and AVG are applied on numeric values
- Function argument may be preceded by DISTINCT

AVG, SUM, MAXIMUN, MINIMUM

- Find the average quantity of all supplied parts.

```
SELECT AVG(QTY) as AvgQ
FROM SP;
```

- Find the total quantity of all supplied parts.

```
SELECT SUM(QTY) as TotQ
FROM SP;
```

- Find the maximum and the minimum weight of all parts.

```
SELECT MAX(WEIGHT) as MaxW,
       MIN(WEIGHT) as MinW
FROM P;
```


GROUP BY queries

- Queries may apply aggregate operators to subsets of rows

Example. For each part get the total number of supplied parts. (Find the total supplied quantity for all supplied parts)

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

SP base table

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

Result

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

Group predicate: HAVING

- When conditions are on the result of an aggregate operator, it is necessary to use the HAVING clause

Example. For each part, with at least 500 supplied pieces, get the total supplied quantity.

```
SELECT P#, SUM(QTY)
 FROM SP
 GROUP BY P#
HAVING SUM(QTY)>=500;
```

SP base table

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

Result

P#	....
P1	600
P2	1000
P4	500
P5	500

Group predicate: HAVING

- Only predicates containing aggregate operators should appear in the argument of the having clause
- Having for the group by is like where for the table rows

Example. Get p# supplied by more than one supplier.

SP base table

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

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

Result

P#
P1
P2
P4
P5

DBMG 37

Exercises (1)

The following relations are given (primary keys are underlined):
JOURNAL (CodJ, NameJ, Editor)
ARTICLE (CodA, Title, Topic, CodJ)

Write the following SQL queries

(a) Find journal names which have published at least an article based on 'motorcycling' topic.
(b) Find journal names which publish motorcycling articles or motor racing articles.
(c) Find journal names which have published at least 2 motorcycling articles.
(d) Find journal names which have published a motorcycling article (only one).

DBMG 38

Solutions (1a)

(a)
```
SELECT DISTINCT NameJ
    FROM JOURNAL, ARTICLE
    WHERE JOURNAL.CodJ=ARTICLE.CodJ
    AND Topic='motorcycling';
```

(b)
```
SELECT NameJ
    FROM JOURNAL, ARTICLE
    WHERE JOURNAL.CodJ=ARTICLE.CodJ
    AND (Topic='motorcycling' OR Topic='motor racing');
```

DBMG 39

Solutions (1b)

(c)
```
SELECT NameJ
  FROM JOURNAL, ARTICLE
  WHERE JOURNAL.CodJ=ARTICLE.CodR AND Topic='motorcycling'
  GROUP BY JOURNAL.CodJ, NameJ
  HAVING COUNT(*)>1;
```

(d)
```
SELECT NameJ
  FROM JOURNAL, ARTICLE
  WHERE JOURNAL.CodJ=ARTICLE.CodJ AND Topic='motorcycling'
  GROUP BY JOURNAL.CodJ, NameJ
  HAVING COUNT(*)=1
```

DBMG 40

Exercises (2)

The following relations are given (primary keys are underlined):
JOURNAL (CodJ, NameJ, Editor)
ARTICLE (CodA, Title, Topic, CodJ)

Write the following SQL queries

(a) Find editors which have published at least an article based on 'motorcycling' topic.
(b) Find editors which publish motorcycling articles or motor racing articles.
(c) Find editors which have published at least 2 motorcycling articles.
(d) Find editors which have published a motorcycling article (only one).

DBMG 41

Solutions (2a)

(a)
```
SELECT DISTINCT Editor
    FROM JOURNAL, ARTICLE
    WHERE JOURNAL.CodJ=ARTICLE.CodJ
    AND Topic='motorcycling';
```

(b)
```
SELECT DISTINCT  Editor
    FROM JOURNAL, ARTICLE
    WHERE JOURNAL.CodJ=ARTICLE.CodJ
    AND (Topic='motorcycling' OR Topic='motor racing');
```

DBMG 42

Solutions (2b)

(c) SELECT Editor
FROM JOURNAL, ARTICLE
WHERE JOURNAL.CodJ=ARTICLE.CodJ AND Topic='motorcycling'
GROUP BY Editor
HAVING COUNT(*)>1;

(d) SELECT Editore
FROM JOURNAL, ARTICLE
WHERE JOURNAL.CodJ=ARTICLE.CodJ AND Topic='motorcycling'
GROUP BY Editor
HAVING COUNT(*)=1

Verbatim search

It is performed by means of *LIKE* constructor
column-name LIKE char-string-const

- _ means any characters
- % means any sequence of n characters

Example. Find all the information relating to parts whose name starts with c.

*SELECT P.**
FROM P
WHERE P.PNAME LIKE 'c%';

P base table

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

Result

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

Verbatim search

- ADDRESS contains the 'Berkeley' string
  ADDRESS LIKE '%Berkeley%'
- S# is exactly of 3 characters and the first one is 'S'
  S# LIKE 'S_ _'
- PNAME is longer or equal to 4 characters and the last fourth is 'c'
  PNAME LIKE '%c_ _ _'
- CITY does not contain 'E'
  CITY NOT LIKE '%E%'

Nested queries

- It is a query nested in an other query
- It is introduced by the IN predicate
- Internal query is executed before than the external query

Example. Find supplier names which supply P2 part.

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

Membership operator

Result

SNAME
Smith
Jones
Blake
Clark

Nested queries

- The last query is equivalent to the following join

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

- For the example database, it is possible to write the query using a predefined set of data

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

Nested queries

Example. Find supplier names which supply at least a red part.

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

Codes of suppliers of red parts

Codes of red products

Result

SNAME
Smith
Jones
Clark

Nested queries

Example. Find codes of suppliers which work in the same city of 'S1'.

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

If we know that the returned value is only one, it is possible to use = or > ... instead of IN.

Result

S#
S1
S4

Nested query

Example. Find s# of all suppliers which have status value smaller than the maximum status stored in the supplier base table.

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

S base table

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

Result

S#
S1
S2
S4

Query with EXISTS

Example. Find the name of suppliers which supply the `P2' part.

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

EXISTS expression (SELECT * FROM ...) is true if and only if the result of the SELECT is different from empty set.

Query with NOT EXISTS

Example. Find the name of suppliers which do not supply the `P2' part.

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

S base table

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 base table

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

Result

SNAME
Adams

Query with NOT IN

Example. Find the name of suppliers which do not supply the `P2' part.

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


Query with NOT EXISTS

Example. Find the name of suppliers with supply all parts.

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

Query with NOT EXISTS

Example. Find the code of suppliers which supply at least all parts supplied by S2.

The search can be divided into steps

- Find all the part codes supplied by the S2 supplier

```
SELECT P#
FROM SP
WHERE S# = 'S2';
```

- With CREATE TABLE and INSERT it is possible to save such data into a TEMP base table
- Next, we search all the suppliers which supply such parts

Query with NOT EXISTS

Example. Find the code of suppliers which supply at least all the product supplied by S2.

```
SELECT DISTINCT S#
FROM SP SPX
WHERE NOT EXISTS (SELECT *
                  FROM TEMP
                  WHERE NOT EXISTS (SELECT *
                                    FROM SP SPY
                                    WHERE SPY.S# = SPX.S#
                                    AND SPY.P# = TEMP.P#));
```


Query with UNION

Given two sets A and B

A UNION B

- is the set of objects such that
- if x belongs to A or belongs to B or belongs to both sets
- x belongs to (A UNION B)
- duplicates are deleted

Query with UNION

Example. Find the code of parts which weight more than 16 or are supplied by S2 supplier, or both events are true.

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


UPDATE

UPDATE table-name
SET column = expression
[, column = expression]
[WHERE condition];

- All table records in *table-name,* which satisfy the condition, are updated according to the *column = expression* in the *SET clause*

Updating of a single record

```
UPDATE P
  SET COLOR = 'Yellow' ,  WEIGHT=WEIGHT+12,  CITY = NULL,
 WHERE P# = 'P1';
```

- The updating is executed only for the record associated to P1 code

Multiple Updating

Example. Update the status of all suppliers in London to double of their current status.

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

- The updating is executed for all records which satisfy the specified condition

Updating with nested query

Example. Set to 0 the quantity supplied by all suppliers in London.

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


Updating of many tables

Example. Update the code of S2 and S9 suppliers.

```
UPDATE S
   SET S# = 'S9'
        WHERE S# = 'S2';
UPDATE SP
   SET S# = 'S9'
        WHERE S# = 'S2';
```

- An UPDATE instruction can update only a table
- There is an integrity problem after the updating of supplier base table
- To guarantee the integrity it is necessary to perform the updating of both tables

DELETE instruction

DELETE FROM table-name
[WHERE condition];

- All records which satisfy the condition are deleted from *table-name*

Example. Delete all supplies.

```
DELETE FROM SP;
```

- SP base table has been emptied

Record deleting

Example. Delete the record corresponding to S1 supplier.

```
DELETE FROM S
       WHERE S# = 'S1';
```

ATTENTION: if SP base table contains reference to S1, the database looses its integrity

Example. Delete all suppliers in Madrid.

```
DELETE FROM S
     WHERE CITY = 'Madrid';
```


Deleting with nested query

Example. Delete all sales by suppliers in London.

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

INSERT instruction

Two cases:

INSERT INTO table-name
[(column [,column]...)]
VALUES (constant [, constant] ...)

INSERT INTO table-name
[(column [,column] ...)]
SELECT ... FROM ... WHERE ... ;

DBMG 67

Inserting of only one record

```
INSERT INTO P (P#, CITY, WEIGHT)
      VALUES ('P7', 'Athens', 24);
```

- A new record is build for P7 part
- NAME and COLOR are initialized to NULL (Attention to CREATE TABLE statement)

DBMG 68

Inserting of only one record

Example. Insert P8 part (name: Sprocket, color: Pink, Weight: 12, city: Nice).

```
INSERT INTO P
      VALUES ('P8','Sprocket','Pink',12,'Nice');
```

- When attribute list is omitted, the statement is equivalent to specify all the attributes according to the creation order of columns in the base table

DBMG 69

Inserting of only one record

Example. Insert a new supply with S20 supplier, P20 part and quantity equal to 1000.

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

ATTENTION: it is necessary than P20 and S20 exist in S and P respectively (integrity problem)

DBMG 70

Inserting of many records

Example. For each part find the code and the corresponding supplied quantity, saving the result into the database.

```
CREATE TABLE TEMP
      (P# CHAR(6),
      TOTQTY INTEGER);

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

DBMG 71

Inserting of many records

- SELECT returns data which are immediately inserted into TEMP base table
- TEMP base table is available for next elaborations
- At the end TEMP table can be deleted

```
DROP TABLE TEMP ;
```

DBMG 72

Integrity constraints in SQL-92

- Constraints are conditions that must be verified by every database instance
  - Intra-relational constraints
  - Inter-relational constraints
  - Assertions

Intra-relational constraints

- Intra-relational constraints involve a single relation
  - not null (on single attributes)
  - unique: permits the definition of keys
    - for single attributes
      - unique, after the domain
    - for multiple attributes
      - unique (Attribute {, Attribute })
  - primary key: defines the primary key (once for each table; implies not null)
  - check

Example of intra-relational constraints

- Each pair of S# and P# uniquely identifies each supply
  - `S# CHAR(6)`
  - `P# CHAR(6)`
  - `unique(S#,P#)`
- Note the difference with the following (stricter) definition which can be used in base tables S and P
  - `S# char(6) not null unique`
  - `P# char(6) not null unique`

Check clause

- It can be used to express arbitrary constraints during schema definition
- Check (*Condition*)
- *Condition* is what can appear in a where clause (including nested queries)

```
CREATE TABLE SP (
      S# CHAR(6) NOT NULL,
      P# CHAR(6) NOT NULL,
      QTY INTEGER
            CHECK(QTY IS NOT NULL AND QTY>0),
      PRIMARY KEY (S#,P#));
```


Inter-relational constraints

- Constraints may take into account several relations
  - references and foreign key permit the definition of referential integrity constraints
    - for single attributes
      - references after the domain
    - for multiple attributes
      - foreign key (Attribute {, Attribute })
      - references ...
  - It is possible to associate reaction policies to violations of referential integrity

Reaction policies

- Reactions operate on the internal table, after changes to the external table
- Violations may be introduced by
  - updates on the referred attribute
  - row deletions
- Reactions
  - cascade: propagate the change
  - set null: nullify the referring attribute
  - set default: assign the default value to the referring attribute
  - no action: forbid the change on the external table
- Reactions may depend on the event

  on < delete | update >

  < cascade | set null | set default | no action >

Example of inter-relational constraint

```
CREATE TABLE SP (
      S# CHAR(6) NOT NULL,
      P# CHAR(6) NOT NULL,
      QTY INTEGER
            CHECK(QTY IS NOT NULL AND QTY > 0),
      PRIMARY KEY (S#,P#),
      FOREIGN KEY (S#) REFERENCES S(S#)
            ON DELETE NO ACTION
            ON UPDATE CASCADE,
      FOREIGN KEY(P#) REFERENCES P(P#)
            ON DELETE NO ACTION
            ON UPDATE CASCADE);
```

DBMG 79

Assertions

- Assertions permit the definition of constraints outside of table definitions
- Useful in many situations (e.g., to express generic inter relational constraints)
- An assertion associates a name to a check clause
  - create assertion *AssertionName* check (*Condition*)

DBMG 80

Example of Assertion

- Constraint: each part can be supplied by at most 10 different suppliers

```
CREATE ASSERTION TooManyS
CHECK (NOT EXISTS
          (SELECT * FROM SP
           GROUP BY P#
           HAVING COUNT(DISTINCT S#)>10))
DEFERRABLE
INITIALLY DEFERRED;
```

DBMG 81

Transactional System

- A system capable of providing
  - the definition and execution of transactions on behalf of multiple
  - concurrent applications

DBMG 82

Transactions

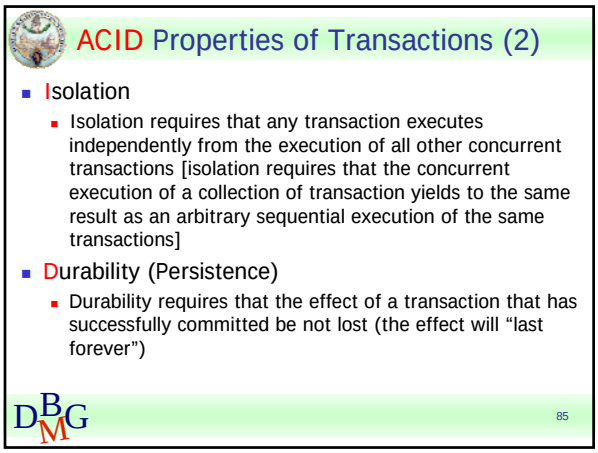
- An elementary unit of work performed by an application, with specific features for what concerns correctness, robustness and isolation
- Each transaction is encapsulated within two commands
  - begin transaction (bot)
  - end transaction (eot)
- Within a transaction, one of the commands below is executed (exactly once)
  - commit work (commit)
  - rollback work (abort)

DBMG 83

ACID Properties of Transactions (1)

- Atomicity
  - A transaction is an atomic unit of work
  - It cannot leave the database in an intermediate state
    - a fault or error prior to commit causes the UNDO of the work made earlier
    - A fault or error after the commit may require the REDO of the work made earlier, if its effect on the database state is not guaranteed
- Consistency
  - Consistency amounts to requiring that the transaction does not violate any integrity constraint
  - Integrity constraint verification can be
    - Immediate: during the transaction (the operation causing the violation is rejected)
    - Deferred: at the end of the transaction (if some integrity constraint is violated, the entire transaction is rejected)

DBMG 84


ACID Properties of Transactions (2)
Isolation
Isolation requires that any transaction executes independently from the execution of all other concurrent transactions [isolation requires that the concurrent execution of a collection of transaction yields to the same result as an arbitrary sequential execution of the same transactions]
Durability (Persistence)
Durability requires that the effect of a transaction that has successfully committed be not lost (the effect will "last forever")
DBMG
85