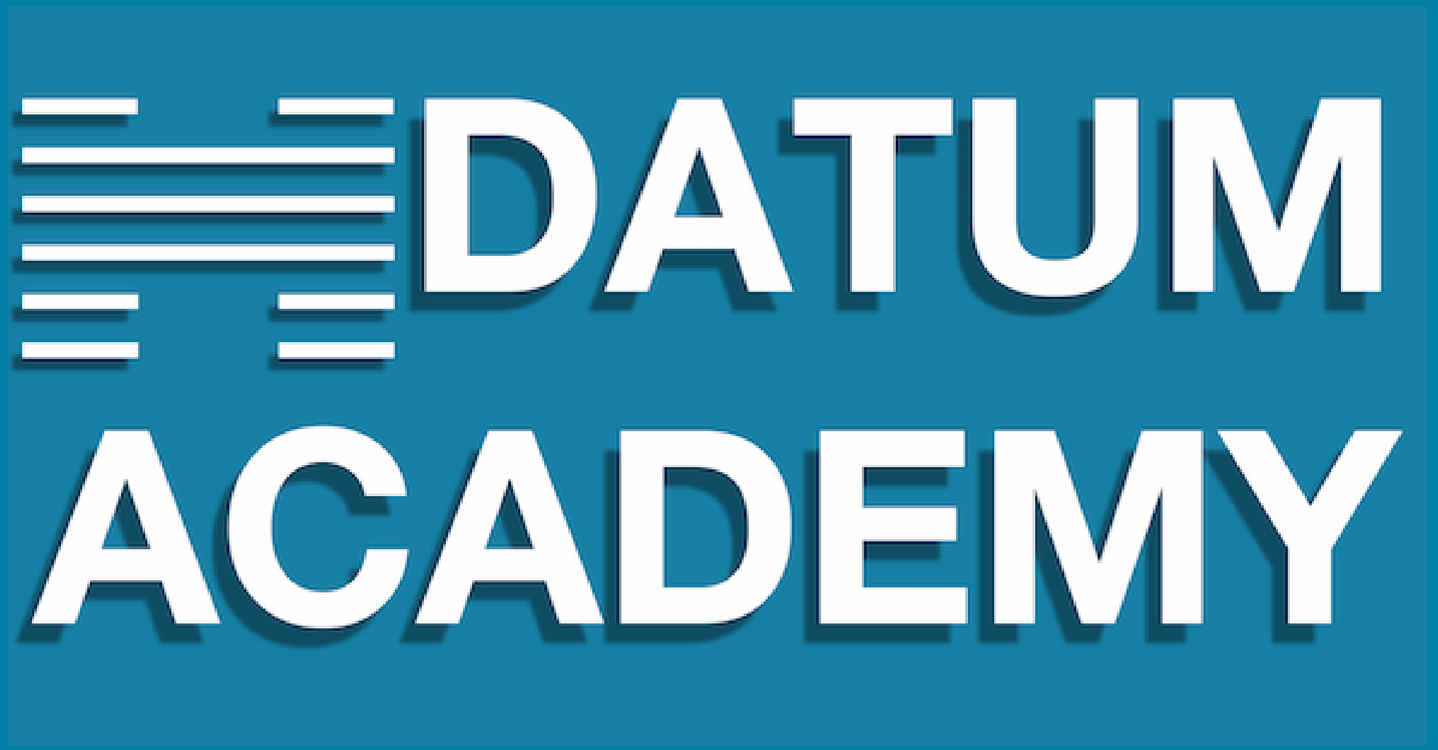


MBDS course :
« From data bases to big data »

(7 lectures)

Professor Serge Miranda
Dept of Computer Science
University of Nice Sophia Antipolis (member of UCA)
Director of MBDS Master degree
(www.mbds-fr.org)

www.mbds-fr.org

Membre de UNIVERSITÉ CÔTE D'AZUR

Introduction to SQL2

(lecture 3)

Professor Serge Miranda
Dept of Computer Science
University of Nice Sophia Antipolis (UCA)
Director of MBDS Master degree
(www.mbds-fr.org)

www.mbds-fr.org

Membre de UNIVERSITÉ CÔTE D'AZUR

SQL standards

SEQUEL (*Structured English as a Query Language*) *from R* System (IBM)

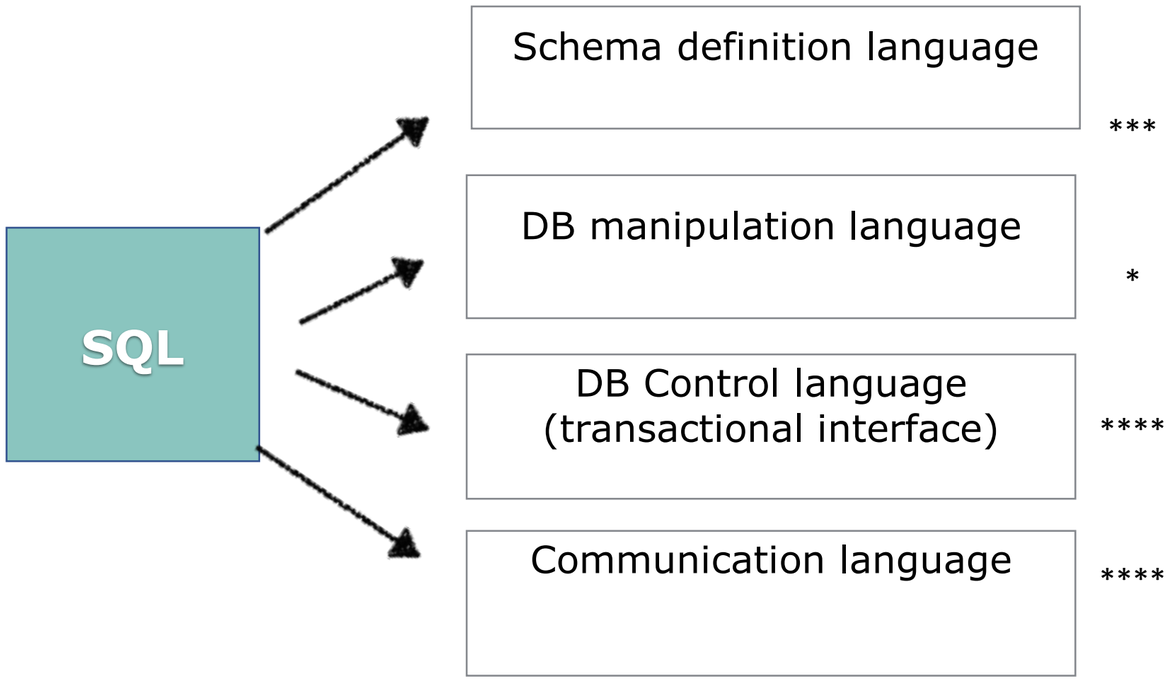
- SEQUEL1
- SEQUEL2 (1976)

---------- **SQL standard in the legacy of SEQUEL** ----------

- SQL1 : ISO (1987) and ANSI (1986)
- **SQL2- 92 (Relational)**
- **SQL3- 99 (Hybrid : Object relational)**
- SQL4-2003 (SQL/OLB Object Language Binding; OLAP)
- SQL5-2006 (SQL/XML ; Xquery)
- SQL6-2008 (SQL/JRT for JaAC functions)
- SQL7-2011 ISO/IEC 9075:2011 Dec 2011

- **SQL8** -? (for Big Data)

The four dimensions of SQL*



*Proceeding from SEQUEL
(Structured English as a QUEry Language)

IBM Research center, San Jose (1975)

Schema definition with SQL2

Relational data structures in SQL

- **Tables (*relations*)**
- **Columns (*attributes*)**
- **Lines (*tuples*)**

PILOT	PIL#	PILNAME	ADDRESS
	1	SERGE	NICE
	2	PETER	NEW YORK
	3	JOEL	DUBLIN

PLANE	P#	PNAME	CAP	LOC
	100	A300	300	Nice
	101	B707	250	NEWYORK
	102	A300	300	DUBLIN
	103	B727	370	LONDON

FLIGHT	F#	PIL#	P#	DC	AC	DT	AT
	IT100	1	100	Nice	Paris	7	8
	IT101	2	100	Paris	Nice	11	12
	IT103	1	103	Lyon	Paris	14	15
	IT104	2	102	Nice	Nice	17	18

SQL mapping with Codd's relational data model

DATUM ACADEMY

Codd's model		SQL (SEQUEL*)
Français	English	
relation	**relation**	**table**
domaine	**domain**	***domain***
attribut	**attribute**	**column**
n-uplet (tuple)	**tuple**	**line**
clé primaire	**primary key**	**primary key**
clé étrangère	**foreign key (primary domain)**	**references**

* SEQUEL : « Structured English as a QUEry Language »

eMBDS

Definition of the relational schema (SQL kernel)

DATUM ACADEMY

Create table

[not null ; primary key / references]

alter table

add [modify, delete]

create / drop index unique

[create /drop data-space]

eMBDS

Example of schema definition (SQL2)

DATUM ACADEMY

Create schema

<Phase1 : Domain creation>

➢**create domain city as char (12)**
default ‘ PARIS ’
check (value in(‘New York ’, ‘ NICE ’, ‘ Dublin ’, ‘Paris’))
create domain Hour as time
check (Value > 7 and Value < 22)

<Phase2 : Relation creation>

➢**create table PILOT**
(PIL# decimal (4),
PILNAME char (12),
ADDR city
check (Value in (‘ Dublin ’, ‘ NICE ’)),
SAL decimal (5),
primary key (PIL#))

eMBDS

Schema definition (Cont')

create table PLANE

P# decimal (4),
PNAME char (12),
CAP decimal (3) check (Value > 100),
LOC city,
primary key (P#))

create table FLIGHT

(F# char(5),
PIL# decimal (4) not null,
P# decimal(4),
DC city,
AC city,
DT hour,
AT hour,
primary key (FLIGHT#),
foreign key (PIL#) references PILOT,
initially deferred,
foreign key (P#) references PLANE,
on delete cascade, on update set null))

alter domain …. Drop constraint

Schema definition (SQL2)

- Syntactic Type :
  - Char(n), decimal, integer, bit, float, date (year, month, day), times (hour, minute, second), timestamp, interval

- **VIEW** : A view is a virtual table (just its definition is stored)

  CREATE VIEW vx AS <SQL Query>

- **Relational catalog** for the schema
  - (INFORMATION-SCHEMA CATALOG)
    containing system tables (SYSTABLE) which could be accessed by SQL

Schema definition : relational catalog

DATUM ACADEMY

Example with DB2 relational catalog.

SYSTABLES	NAME	CREATOR	COLCOUNT
	PILOT	Serge	4
	PLANE	Serge	4
	FLIGHT	Serge	7

SYSCOLUMNS	NAME	TBNAME	COLTYPE
	PIL#	PILOT	SMALLINT
	PILNAME	PILOT	ChaR
	ADDR	PILOT	ChaR
	P#	PLANE	SMALLINT
	…	…	…

SYSINDEX	NAME	TBNAME	CREATOR
	PX	PILOT	Serge
	AX	PLANE	Serge
	VX	FLIGHT	Serge

eMBDS

Schema definition : clusters

Cluster : physical JOIN

10	SERGE	NICE					
	IT 100	10	Paris	100	Nice	7	8
	IT 101	10	Nice	101	Paris	11	12
	IT 105	10	Paris	104	Toulouse	15	16

Example : Cluster PF between PILOT and FLIGHT on PIL#

Here cluster block for PIL# = 10

Create cluster PF (PIL# number (4))

(Cluster key)

Schema definition : cluster

Clustering within CREATE TABLE

- Example :

```
create table PILOT
      (PIL#  number(4) primary key, …)
      cluster PF (pil#)
```

Clustering an existing table

- Example :

```
create table FLIGHT2 cluster PF (PIL#)
      as select * from FLIGHT
      drop table FLIGHT
      rename FLIGHT2 to FLIGHT
```

- NOTE : We can
  - Eliminate a table from a cluster :
    alter cluster PF drop table PILOT
  - Delete a cluster :
    drop cluster PF

DATUM ACADEMY

DB manipluation with SQL

Data retrieval (SELECT) and storage operators (INSERT, UPDATE, DELETE)

eMBDS

DB interaction

DATUM ACADEMY

➢ 1st mode : *interactive SQL*

<Query mapping with SQL>

Select <target list or «*»>
From <tables (or *views*) >
Where <expression on lines>
with in, exists, any, all, and, or, not, between, like, …
group by <partitionned attributes>
Having <expression on partitions>
Order <tri >
Union

eMBDS

SQL manipulation : storage operators

DATUM ACADEMY

UPDATE :

```
update    R
set
where
```

DELETE :

```
delete
From    R
where
```

INSERT :

```
insert
into    R
Values
```

Note : heterogeneous syntax ☹

eMBDS

DB interaction

2nd mode : *embedded SQL* (in programming language)

- **impedance mismatch :**

SAT (Set at-a-time)and RAT (Record at–a-time)

- Logical pointer : **CURSOR** (« impedance mismatch »)

(exec sql)
declare CX cursor for < SQL query >

(exec sql)
open / fetch / update / delete / close CX (cursor)

Current line

sqlcode

SQL2 by example

Domain : =
{Nice, Paris, London,Dublin }

PILOT	PIL#	PILNAME	ADDR
	100	Serge	Nice
	101	Jean	Paris
	102	Paul	Paris
	…	…	…

FLIGHT	FLIGHT#	PIL#	AC…
	IT200	102	Nice
	IT201	100	Paris
	IT204	102	Paris

Basic SQL mapping : SELECT-FROM-WHERE

DATUM ACADEMY

select - from - where ?

PLANE	AVN	PNAME	CAP	LOC
	100	A300	300	Nice
	101	B707	250	Paris
	102	A300	300	London
	103	B707	200	Nice

PLANE	PNAME	CAP
	A300	300
	B707	200

```
Select    PNAME, CAP
from      PLANE
where     LOC = ` NICE ' ;
```

In relational algebra ?

eMBDS

SQL by example

Every line from FLIGHT (complete table) ?

```
SELECT *
FROM FLIGHT
```

SQL query : interactive and embedded mode

DATUM ACADEMY

What are the flights arriving in Paris in the afternoon ?

- **interactive mode :**

```
select * from FLIGHT   where AC = ‘ PARIS ’ and DT > 12 ;
```

- **Embedded mode :**

```
exec sql declare CX cursor for
select FLIGHT#,DT,AT
from FLIGHT where AC = ‘ PARIS ’ and DT > 12;

declareFLIGHT char(5) ; DT date ; AT date ;
exec sql open CX ;
do
exec sql fetch CX into : FLIGHT, DT, AT ;
<variable processing>
end
exec sql close CX ;
```

eMBDS

Exercice : On DB2 Catalog use SQL to answer the following queries

DATUM ACADEMY

- *What are the names of the columns of FLIGHT table ?*
- *What are the tables of the schema ?*
- *What are the tables which contain PIL# column ?*

eMBDS

Solution

Relational catalog (in DB2) with eleven tables. Here we ll use :

- SYSTABLE (TBNAME, NBCOLUMNS, PK,...)
- SYSCOLUMNS (TBNAME, CNAME, Type,...)

Solution with SQL

DATUM ACADEMY

- *What are the names of the columns of FLIGHT table ?*

```
Select    CNAME
from      SYSCOLUMNS
where     TBNAME = ‘ FLIGHT ’ ;
```

- *What are the tables of the schema ?*

```
Select    TBNAME
from      SYSTABLES ;
```

- *What are the tables which contain PIL# column ?*

```
Select    TBNAME
from      SYSCOLUMNS
where     CNAME = ‘ PIL# ’ ;
```

eMBDS

JOIN query with SQL

DATUM ACADEMY

- ***What are the names of the pilots which are insuring a flight towards Paris ?***

2 basic versions for SQL (which is an hybrid query language) : PREDICATIVE and SET–oriented

Predicative version

```
select    PILNAME
from      PILOT, FLIGHT
where     PILOT. PIL# = FLIGHT. PIL# < join predicate>
          and FLIGHT. AC = ‘ PARIS ’ ;
```

eMBDS

SET-oriented JOIN with SQL (example)

DATUM ACADEMY

SET-oriented version (with subquery)

```
select     PILNAME
from       PILOT,
where      PIL#  IN
        (Select   PIL#
        From      FLIGHT
        Where     FLIGHT. AC = ‘ PARIS ’) ;
```

Other JOINs in SQL

- NATURAL JOIN in the FROM clause (PK and FK implicitly)

  Example :

  ```
  select *
  from (PILOT natural join FLIGHT) as PV ;
  ```

- Replace « IN » by « = ANY »

- Use « EXISTS (Select *…) » <existential quantifier for Join>

GROUP BY in SQL : partitioning

- GROUP BY does not exist in the relational algebra !
- Example :

select PIL# from FLIGHT group by PIL# :

FLIGHT	FLIGHT#	PIL#	P#	DC	AC
	IT100	1	50	NIce	Paris
	IT101	2	50	Paris	Toulouse
	IT103	2	50	Toulouse	Paris
	IT104	1	51	Paris	Nice
	IT105	1	52	Nice	Lyon

group by PIL#

PIL#=1	FLIGHT#	P#	DC	AC
	IT100	50	Nice	Paris
	IT104	51	Paris	Nice
	IT105	52	Nice	Lyon

PIL#=2	FLIGHT#	P#	DC	AC
	IT101	50	Paris	Toulouse
	IT103	50	Toulouse	Paris

Basic aggregate functions with Group BY or SELECT/WHERE :
MAX, MIN, SUM, AVG, COUNT

HAVING clause (to discard partitions)

On the previous example :

```
Select PIL#, COUNT(*) …
from FLIGHT
group by PIL#
Having count (*)  > 2 ;
```

< ➔ eliminates the partition corresponding to PIL#=2 >

Exercice

- For every city where a plane is located give the average of the capacities of planes located there ?
- What is the number of FLIGHTs ?
- What is the average number of the planes located in Nice ?

Solution

DATUM ACADEMY

For every city where a plane is located give the average of the capacities of planes located there ?

```
SELECT    LOC, AVG (CAP)
FROM      PLANE
GROUP     BY LOC
```

What is the number of FLIGHTs ?

```
Select    COUNT (*)
From      FLIGHT;
```

What is the average number of the planes located in Nice ?

```
select    avg (CAP)
from      PLANE
where     LOC = ' NICE ' ;
```

eMBDS

Exercice :
from SQL to English for ?

DATUM ACADEMY

Q1 :

```
Select PIL#, COUNT(*)
from FLIGHT
group by PIL#
Having count (*)  > 2 ;
```

Q2 :

```
select PIL#, count (*)
from FLIGHT
where DC = ‘ NICE ’
group by PIL#
Having count (*) > 2 ;
```

Q3 :

```
Select PIL#, count(*)
from FLIGHT
where PIL#                      (2)
      in (select PIL#
      from FLIGHT               (1)
      where DC = ‘ NICE ’)
group by PIL#                   (3)
Having count (*) > 2 ;          (4)
```

eMBDS

Solution

Q1 SQL :

```
select PIL#, count (*)
from FLIGHT
group by PIL#
Having count (*)  > 2 ;
```

English query :

« What are the numbers of the pilots who insure more than two flights (with the number of flights they insure) ? »

Q2 SQL :

```
select PIL#, count (*)
from FLIGHT
where DC = ‘ NICE ’(1)
group by PIL#  (2)
Having count (*) > 2 ;    (3)
```

English query :

« What are the numbers of the pilots who insure more than two flights FROM NICE (with the number of flights they insure) ? »

Solution

Q3 SQL :

```
Select PIL#, count(*)
from FLIGHT
where PIL#                    (2)
      in (select PIL#
      from FLIGHT             (1)
      where DC = ‘ NICE ’)
group by PIL#                 (3)
Having count (*) > 2 ;              (4)
```

English query :

What are the numbers of the pilots who insure more than two flights WITH AT LEAST ONE FROM NICE (with the number of flights they insure) ?

Synonym variables in SQL query

We can define a synonym variable for any table in the FROM clause which then will be used as a prefix of any attribute in a SELECT or WHERE Clause. This could be done for convenience reasons but it is mandatory to use them with dependent subqueries

Example : What are the planes which are in service from Nice ?

```
select *
from PLANE (as) AVX
where AVX. P# in
        (select VY. P#
        from FLIGHT (as) VY
        Where VY.DC = 'Nice') ;
```

Synonym variables

- *What are the planes whose capacity is more than 10% greater than the average of plane capacities ?*

```
select *
from PLANE
where CAP >
      (select avg (CAP)* 1.1
      from PLANE) ;
```

Synonym variables

DATUM ACADEMY

- *What are the planes whose capacity is more than 10% greater than the average of the capacities* ***of the planes located in the same city ?***

```
Select *
from PLANE AX
where AX.CAP >
    (select avg (AY.CAP)* 1.1
    from PLANE AY
    where AX.LOC = AY.LOC);
```

Synonym variables are mandatory to avoid ambiguity in dependent sub queries

Storage operators in SQL

Q : Delete flights from Nice ?

```
delete FLIGHT
where DC = ' NICE ' ;
```

Q : *Insert tuple (1, Jean Paris) into PILOT*

```
Insert Into PILOT
VALUES (« 1 », « Jean », « Paris »);
```

Q : *Add 10 to the capacity of Airbus located in Nice* ?

```
update PLANE
set CAP = CAP + 10
where PNAME = ' AIRBUS ' and LOC = ' NICE ' ;
```

Why not a consistent syntax ?

DATUM ACADEMY

SELECT/INSERT/UPDATE/DELETE
From <Table>
Where <qualification; **. . .**>

eMBDS

Exercice : Update

- Write the following update query in SQL :

Increase by 10 the capacity of the AIRBUS in service from NICE and conducted by pilots living in PARIS ?

Solution

Increase by 10 the capacity of the AIRBUS in service from NICE and conducted by pilots living in PARIS ?

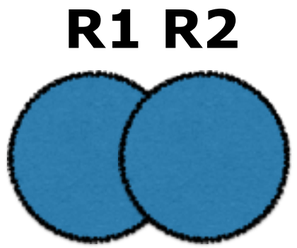
```
update PLANE
set CAP = CAP + 10
where PNAME =  ‘ AIRBUS ’ and
P# in (select P#
        from FLIGHT
        where DC = ‘ NICE and
        PIL# in (select PIL#
                 from PILOT
                 where ADDR = ‘ PARIS ’)) ;
```

Exercices :
Codd’s algebra and SQL2

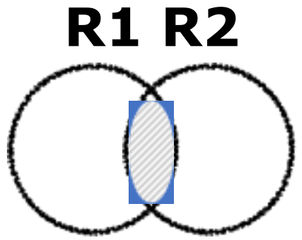
(cf CODD’s theorem) !

SET OPERATORS and SQL

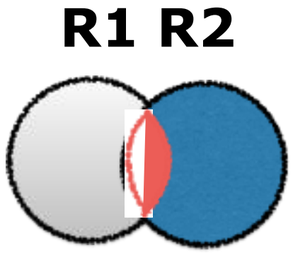
- SET OPERATORS with VENN's diagrams
- Relations should be ***union compatible :***
  - same number of attributes
  - corresponding attributes are defined on the same domain



UNION



INTERSECTION



DIFFERENCE

RELATIONAL ALGEBRA and SQL

DATUM ACADEMY

- Relational operators in SQL2

  Tables PILOT, PLANE, FLIGHT with two views :

  Create view PNICE as (select * from PLANE where LOC = ' NICE ') ;

  Create view PAIRBUS as (select * from PLANE where PNAME = ' AIRBUS ') ;

- SET operators in SQL : UNION, INTERSECT, EXCEPT (DIFFERENCE)

  Example :

```
select P#
from PNICE

union

select P#
from PAIRBUS ;
```

Relational operators in SQL

DATUM ACADEMY

INTERSECTION

SQL1	SQL2
select P# from PNice	(select P# from PNice)
where exists	**intersect**
(select P# from PAIRBUS.P#= PNICE.P#);	(select P# from PAIRBUS);

DIFFERENCE

SQL1	SQL2
select P# from PNice	(select P# from PNice)
where not exists	**except**
(select P# from PAIRBUS.P#=PNICE.P#);	(select P# from PAIRBUS);

eMBDS

Relational operators in SQL

DATUM ACADEMY

SELECTION

```
select *
from PLANE
where CAP > 200 ;
```

PROJECTION

```
select P#, PNAME
from PLANE.
```

Relational operators in SQL

DATUM ACADEMY

JOIN

What are the names of the pilot in service from NICE?

Many solutions for JOIN

Here the predicative one

```
select PILNAME
from PILOT, FLIGHT
where FLIGHT.DC = ' NICE '
and PILOT.PIL# = FLIGHT.PIL# ;
<join predicate>
```

The SET version :

```
Select PILNAME
From PILOT where PIL#IN (Select PIL# From
FLIGHT where DC ='NICE');
```

DIVISION

What are the numbers of the pilots who are driving EVERY PLANE ?

With double negation and synonym variable <see later>

```
select (distinct) VX.PIL#
from FLIGHT VX
where not exists
    (select *
    from PLANE
    where not exists
        (select *
        from FLIGHT VY
        where VY.PIL# = VX.PIL#
        and PLANE.P# = VY.P#));
```

eMBDS

4 other joins

DATUM ACADEMY

```
select PILNAME
from PILOT
where PIL# = ANY
      (select PIL#
      from FLIGHT
      where FLIGHT.DC = ' NICE ') ;
```

```
select PILNAME from PILOT where exists
(select *
from FLIGHT
where PILOT.PIL# = FLIGHT.PIL# and
FLIGHT.DC = ' NICE ') ;
```

```
select PILNAME
from PILOT
where 0 < ( select count (*)
      from FLIGHT
      where PILOT.PIL# =FLIGHT.PIL# and
      FLIGHT.DC = ' NICE ') ;
```

```
select PILNAME
from PILOT
where ' NICE  ' = any (select DC from
FLIGHT where FLIGHT.PIL# = PILOT.PIL#) ;
```

eMBDS

Other JOINs in SQL2

DATUM ACADEMY

- **NATURAL JOIN**

Example :

```
select PILNAME
from PILOT (NATURAL) join FLIGHT on FLIGHT.DC = 'NICE'
[using PILOT.PIL# = FLIGHT.PIL# ;] <Primary key and foreign key>
```

- **OUTER JOIN (LEFT, RIFHT, FULL)**
  - EXAMPLE **:**

```
FROM PILOT left join FLIGHT using PIL# ;
FROM PILOT natural left join FLIGHT
FROM PILOT left join FLIGHT on PILOT.PIL#  = FLIGHT.PIL# ;
```

- **UNION JOIN, CROSS JOIN** (cartesian product)
- **A join B on C using att1** corresponds to :

```
select * from A , B where C and A. att1 = B. att1 ; <join predicate>
```

eMBDS

DIVISION in SQL (double negation)

$$\forall X\, P(X) \equiv \neg\exists\, X \neg P(X)$$

NOTE : In SQL : $\neg\exists$ is written NOT EXISTS (SELECT *..)

DIVISION in SQL (double negation)

- *Select (all) the names of the pilots who are driving EVERY AIRBUS of the company ?*

➡ (double negation) and the following equivalent query to be processed in SQL :

- *Select the names of the pilots such as IT DOES NOT EXIST AIRBUS in the company WHICH IS NOT DRIVEN by those pilots*

DIVISION
with double negation in SQL

DATUM ACADEMY

Select the names of the pilots such as

IT DOES NOT EXIST AIRBUS in the company

WHICH IS NOT DRIVEN by those pilots

```
select PILNAME
from PILOT
    where not exists
    (select *
    From PLANE
    where PLANE.PNAME = ' AIRBUS '
        and not exists
        (select*
        from FLIGHT
        where PLANE.P# = FLIGHT.P#
        and PILOT.PIL# = FLIGHT.PIL#));
```

eMBDS

DIVISION With CONTAINS *(SEQUEL)*
<SET INCLUSION>

DATUM ACADEMY

SELECT the names
of the pilots
such as

```
select PILNAME
from PILOT
Where
```

The set of planes conducted
by these pilots

```
(select P#
from FLIGHT
where FLIGHT.PIL# = PILOT.PIL#)
```

INCLUDES

CONTAINS

the SET of AIRBUS
(ALL)

```
(select P#
from PLANE
where PNAME = ‘ AIRBUS ’);
```

eMBDS

DIVISION with GROUP BY and COUNT (Oracle) inspired by SEQUEL

We count the number of (different) Airbus driven by the pilots and the number of Airbus in the company; if they match then these pilots are driving every Airbus

```
Select PIL#
from FLIGHT, PLANE, PILOT
where PILOT.PIL#= FLIGHT.PIL# and PLANE.P#= FLIGHT.P# and PNAME= 'Airbus'
Group by PIL#
Having count (distinct P#) = <in the subquery* the number of Airbus in the
Company (ALL)>
    (select count (*)
    from PLANE
    where PNAME = ' AIRBUS ');
```

** Such subquery with a function might not be authorized in some SQL implementation*

DATUM ACADEMY

DB CONTROL with SQL

VIEWS and TRANSACTIONS

eMBDS

SQL DB Control : VIEWS

- **CREATE VIEW vx <AS SQL query>**
  - Virtual table (only its definition is stored)
  - Example on PILOT (PIL#, PILNAME, ADDR, SALARY) let us define the *Privacy view LOW-SALARY for pilots earning less than 4000* ? :

```
create view BAS-SALAIRE as
        (select * from PILOT
        where SAL < 4.000) ;
```

- **GRANT / REVOKE rights on views**
  - Example : **Grant update on BAS-SALAIRE to SERGE**

SQL DB control : TRANSACTION

TRANSACTION : Cohort of (SQL) operators satisfying the ACID properties i.e.maintaining a consistent state of the data base in front of concurrency or crash

The cohort of operators is ATOMIC *(« All or nothing »)* and enables to go from one consistent state of the DB to another or remain in the same state

ACID Properties of Transactions

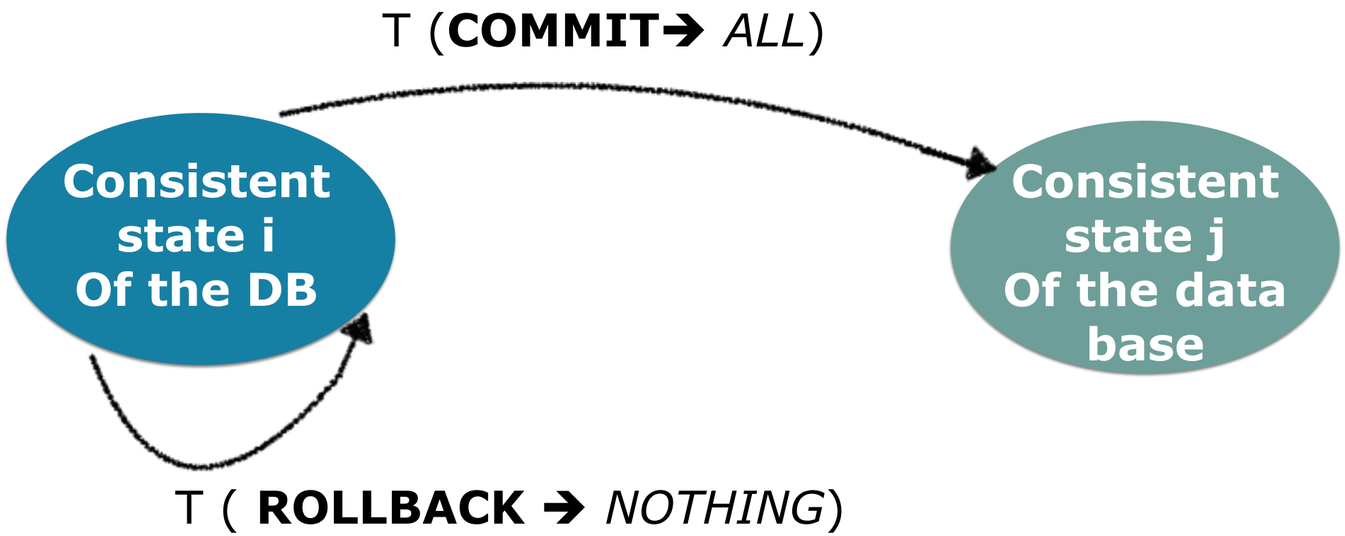
DATUM ACADEMY

Two DB consistency issues solved with TRANSACTION concept

- **CRASH RECOVERY**
  - **A**tomicity (COMMIT/ROLLBACK)
  - **C**onsistency
- **INTERFERENCE of concurrent operations**
  - **I**solation < LOCKING>
  - **D**urability

eMBDS

SQL Transaction verbs



In SQL2, beginning of a transaction is implicit (BEGIN TRANSACTION in some implementations). Two modes for termination : normal (COMMIT) or abnormal (ROLLBACK)

SQL verbs for transactions

DATUM ACADEMY

begin / end transaction

Commit (Work) <Save Point>

Rollback (work)

lock table in exclusive mode or shared mode

DB Control (Transaction)

- ***Atomicity*** ➔ **FAULT/CRASH TOLERANCE**
  - **ALL or NOTHING :**
    - **Either back recovery to the beginning of the transaction (or SAVE POINT)**
    - **or full completion of the transaction (ALL)**
- ***SAVEPOINT***
  - **Unit of consistency**
  - **Limited rollback work**
  - **Structured transaction programming**
    - **Example :**

```
insert into PASSENGER
Values (1, ' serge ', +33 6060606) ;
savepoint after insert ;… rollback  after inser t ;… commit
```

Well-Formed serializable transactions

- **Serializibility → Transaction concurrency**
  - Parallel execution is equivalent to its serial execution
  - (each stand-alone transaction is ACID)
- ***Well formed* transaction**
  - Setting a SHARE LOCK before reading (Select)
  - Setting an EXCLUSIVE lock before a updating
- ***Two-phase* locking**
  - LOCK phase (lock increase)
  - followed by UNLOCK phase (without interference)

JIM GRAY's theorem

« Every well-formed transaction with two-phase locking is serializable »

DATUM ACADEMY

Exercices

ORDER clause

- *Q1 : What are the names of the planes with their number and location (other than Nice) whose capacity is greater than 2000 with a decreasing order on their number ?*

```
select P#, PNAME, LOC
from PLANE
where CAP > 200 and LOC≠ ‘ NICE ’
order by P# dsc ;
```

<dsc : descending vs asc : ascending>

JOIN among 3 relations

DATUM ACADEMY

- Q2 : What are the names of the pilots driving an A320 ?

SET- mode : (A)

```
select PILNAME
from PILOT
where PIL# in
        (select PIL#
        from FLIGHT
        where P# in
                (select P#
                from PLANE
                where PNAME = ' A320'));
```

Join among 3 relations

DATUM ACADEMY

- Q2 : *What are the names of the pilots driving an A320 ?*

Predicate mode : (A)

```
select PILNAME
from PILOT, PLANE, FLIGHT
where PILOT. PIL# = FLIGHT.PIL# and FLIGHT.P# = PLANE.P#
and PLANE.PNAME = ‘ A320 ’;
```

Join among 3 relations

DATUM ACADEMY

- Q2 : *What are the names of the pilots driving an A320 ?*

Hybrid mode : (A)

```
Select PILNAME
From PILOT
Where PIL# in
        (Select PIL#
        From FLIGHT, PLANE
        Where FLIGHT.P#= PLANE.P#
        and PLANE.PNAME = 'A320');
```

Joins between non-key attributes

DATUM ACADEMY

- Q3 : *What are the names of the pilots who LIVE in a city where an A320 is located ?*

(B)

```
select PILNAME
from PILOT, PLANE
where PILOT. ADDR = PLANE. LOC
and PLANE.PNAME = ' A320 ';
```

Logical queries

- UNION, INTER, MINUS (difference) corresponding to OR AND, NOT

Example : Q3 ’ = Q3 and Q2 :

What are the names of the pilots driving an A320 and who live a city where an A320 is located ?

➔ RESULT = (A) inter (B) <see previous queries>

Joins on multiple attributes

- *Q4 : What are the flights whose trip (DC, AC) is identical to those insured by Serge ?*

```
select *
from FLIGHT
where (DC, AC) in
        (select DC, AC
        from FLIGHT, PILOT
        where FLIGHT. PIL# = PILOT.PIL#
        and PILOT.PILNAME = ‘ Serge ’);
```

Multiple join

DATUM ACADEMY

- *Q5 : What are the pilots (PILNAMES) living in a city where an A320 is located and who are on duty from a city (DC) whose arrival city (AC) is also served by SERGE ?*

```
select PILNAME
from PILOT
where ADDR in (select LOC from PLANE where PNAME =‘ A320 ’)
and PIL# in (
        select PIL#
        from FLIGHT
        where DC in (
                select AC
                from FLIGHT, PILOT
                where FLIGHT.PIL# = PILOT.PIL# and PILOT.PILNAME =
‘ Serge ’));
```

eMBDS

Exercice :
Q5 in full SQL predicate form ?

DATUM ACADEMY

- *Q5 :What are the pilots living in a city where an A320 is located and who are on duty from a city (DC) whose arrival city (AC) is also served by SERGE ?*

eMBDS

Q5 in predicate form

- What are the pilots living in a city where an A320 is located and who are on duty from a city (DC) whose arrival city (AC) is also served by SERGE ?

- In predicate form with synonym variables

Select P1. PILNAME
From PILOT P1 P2, Plane, Flight F1 F2
Where P1.ADDR- Plane.loc and
P1.pil#= F1.pil# and
F1.DC= F2. AC and
F2.pil#= P2.pil# and
P2.pilname =‘serge’;

ANY and ALL* in SQL

DATUM ACADEMY

- *Q6 : What are the planes whose capacity is greater than every capacity of planes located in NICE ?*

```
select * from PLANE
where CAP > all (select CAP  from PLANE where LOC = ' NICE ') ;
```

- Q7 *: What are the planes whose capacity is greater than (at least) the capacity of a plane located in NICE ?*

```
select *
from PLANE
where CAP > any (select CAP from PLANE where LOC = ' NICE ');
```

*Not to be confused with ALL the special value in the CUBE operator (like NULL)

* Do not confuse it with the ALL value of the cube operator

eMBDS

Multiple result

- Q8 : *Give all the pairs of pilots (pil#) living in the same city ?*

```
select PIL1.PIL#, PIL2.PIL#
from PILOT PIL1 PIL2
where PIL1.ADDR = PIL2.ADDR and PIL1.pil# ≠PIL2.pil#
```

Synonym variable

- *Q9 : What are the pilots who are driving a plane driven also by a pilot living in the same city ?*

```
select P1.*
from PILOT  P1, FLIGHT  F1
where  P1.PIL# = F1.PIL#
and F1.P#  in
(select F2.P#
from FLIGHT F2, PILOT P2
Where F2.pil#= P2.pil# and P2.ADDR = P1.ADDR and
P1.pil#≠ P2.pil#)
```

GROUP BY

DATUM ACADEMY

- Q10 : *For every pilot in service count their number of flights they insure*.

```
select PIL#, count (*)
from FLIGHT
group by PIL#
```

- Q11 : For every plane in service during the afternoon and driven by more than 2 pilots, what are the concerned trips (DC,AC) ?

```
select P#, DC, AC
from FLIGHT
where DT > 12
group by P#
having count (PIL#) > 2 ;
```

Aggregate SQL2 functions

Basic SQL2 aggregate functions : AVG, SUM, MIN, MAX, COUNT.

- Q12 : *What is the number of trips (DC, AC) in the afternoon ?*

```
select count distinct (DC, AC)
from FLIGHT
where DT > 12 ;
```

Exercice

- Q13 : *What are the pairs of cities such as a plane located in the first city is driven by a pilot living in the second ?*

Exercice

DATUM ACADEMY

- Q13 : *What are the pairs of cities such as a plane located in the first city is driven by a pilot living in the second ?*

```
select PLANE.LOC, PILOT.ADDR
from PILOT, PLANE, FLIGHT
where PILOT.PIL# = FLIGHT.PIL#
and PLANE.P# = FLIGHT.P# ;
```

outer join

DATUM ACADEMY

- *Q14 : For each pilot give his address and for those who are in duty the list of flights they insure ?*
- SQL writing to understand the meaning of the outer join :

```
select PILOT.PIL#, PILOT.ADDR, FLIGHT.FLIGHT#
From PILOT, FLIGHT
where PILOT.PIL# = FLIGHT.PIL#
union
select PILOT.PIL#, PILOT.ADDR, ' bb '
from PILOT
where not exists
      (select *
      from FLIGHT
      where PILOT.PIL# = FLIGHT.PIL#);
```

Note : in SQL there exists a direct JOIN operator to do it : OUTER JOIN (LEFT, RIGHT, ALL)

eMBDS

Exercice

DATUM ACADEMY

- **New relation in the schema :**

TRAINING (PIL#, PLANETYPE, DATE)

Q15 *: What are the maximum and average capacities of Airbus (LIKE AIRBUS%) in service from NICE and driven by PILOTS whose name sounds like SMITH (SOUND 'SMITH') who received a training and drive them from cities whose number of located planes is equal to the number of planes located in Nice ?*

Exercice

DATUM ACADEMY

Q15 *: What are the maximum and average capacities of Airbus (LIKE AIRBUS%) in service from NICE and driven by PILOTS whose name sounds like SMITH (SOUND 'SMITH') who received a training and drive them from cities whose number of located planes is equal to the number of planes located in Nice ?*

```
SELECT LOC, MAX (Cap), AVG (Cap)
FROM Plane, Flight, Pilot
Where Plane.p#= Flight.p# and Flight.pil#= Pilot.pil# and PNAME LIKE
'AIRBUS%' and PILNAME SOUND ('SMITH') and Flight.DC = Plane.LOC and
Pil# in (Select pil# from Training where planetype = Pname)
GROUP BY LOC
HAVING COUNT (Plane.loc) = (SELECT COUNT (*) FROM PLANE WHERE LOC =
'NICE') ;
```

eMBDS

Exercice 1 (Division in SQL2)

STUDENT (S#, SNAME, DEGREE)
COFFEEHOUSE (C#, LOC),
INVITATION (S1TING, S2TED, C#, Date, Itype)

- *What are the students who invited MARIA from MBDS degree to have a chocolate in every coffee house of Vieux Nice ?*

Double negation

« …such it does not exist any Coffeehouse in Vieux-Nice where the student did not invite Maria ? »

Select S1.S #
From Student S1
Where
NOT EXISTS *<1st negation : DIVISOR : « ALL Coffee Houses » in Vieux Nice>*
(SELECT *
From COFFEEHOUSE
Where LOC = 'VIEUX-NICE' **and**
NOT EXISTS *<2nd negation with double link>*
(SELECT *
From Invitation, Student Sx
Where Invitation.C #= Coffeehouse.C # and and Itype = 'Chocolate' and S1.S # = Invitation.S2TING and Invitation.Sted =sx.S# and sx.sname= 'Maria' and sx.degree = 'MBDS';

Exercice

RENTACAR (RC#, City, Numbercars)

CUSTOMER (CL#, CLNAME, CLADDRESS, Company)

CAR (C#, CNAME, Model, Power, Color)

RENTAL (RC#,CL#,C#,DateRES, Duration, DD, DR)
<DD : date departure; DR : Date Return>

Give a schema definition in SQL2 (CREATE TABLE)

Give an example of an SQL Join in three different types ?

Write the SQL query (three modes) and the relational algebra for the following retrieval :

« *What are the Oracle customers who rent every Red Honda car in Nice Rentacar agency ?* »

In Codd's algebra

- ***« What are the Oracle customers who rent every Red Honda car in Nice Rentacar agency ? »***

- **DIVISOR : « every red Honda car »**

V1 = SELECT CAR (Model = 'Honda' AND Color = « red »)

V2 = PROJECT V1 (C#)

- **DIVIDEND : (CL#, C#)**

A1 = SELECT RENTACAR (city = 'Nice')

C1 = JOIN RENTAL (RC# = RC#) A1

C2 = SELECT CUSTOMER (COMPANY = 'Oracle')

C3 = JOIN C1 (CL# = CL#) C2

DD = PROJECT C3 (CL#,C#)

RESULT = DD/V2

In SQL2 and SEQUEL

DATUM ACADEMY

- *« What are the Oracle customers who rent every Red Honda car in Nice Rentacar agency ? »*
  - ***DOUBLE NEGATION***

    What are the Oracle customers ***so it does not exist any Red Honda car that they did not rent in Nice ?***

```
SELECT R1. CL#
FRom RENTAL R1, CUSTOMER, RENTACAR
Where R1.CL#= customer.CL# and RENTACAR. RC#= R1. RC# and rentacar.city = 'Nice' and
    customer.company= 'Oracle'
    And NOT EXIST
    (SELECT *
    From CAR
    Where Model= 'Honda' and Color = 'red'
            And NOT  EXIST
            (Select *
            From RENTAL R2
            Where R2.CL#= R1.CL# and R2.C# = CAR.C#));
```

SEQUEL

➢SEQUEL

SELECT CUSTOMER.CL# *<could be taken in RENTAL or CUSTOMER>*
From RENTAL, CUSTOMER, RENTACAR
Where RENTAL.CL# = Customer.CL# and RENTACAR.RC# = RENTAL.RC# and RENTACAR.CITY= 'Nice' and Customer.Company = 'Oracle' and
(SELECT RENTAL.C# *<set of cars rent by customers of the result>*
From RENTAL
Where RENTAL.CL#= Customer.CL#)
EQUAL/CONTAINS *<divisor>*
(Select C#
From CAR
Where Model = 'Honda' and Color = 'red') ;

With GROUP BY and COUNT

SELECT CUSTOMER.CL# *<could be taken in RENTAL or CUSTOMER>*

From RENTAL, CUSTOMER, RENTACAR, CAR

Where RENTAL.CL# = Customer.CL# and RENTACAR.RC# = RENTAL.RC# and RENTACAR.CITY = 'Nice' and Customer.Company = 'Oracle' and car.C# = Rental.c# and model = 'Honda" and Color = 'red'

GROUP BY CL#

*HAVING Count (C#) = (Select COUNT * From Car where Model = 'honda' and color = 'red') ;*

Some Books on SQL

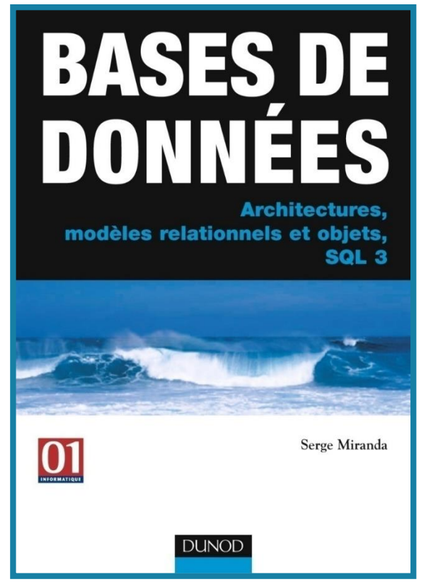
English :

- **Chris Date « An Introduction to data base systems » (8th Edition), Addison Wesley**
- E.F Codd (1990). « *The Relational Model for Database Management* » (Version 2 ed.). Addison Wesley Publishing Company. ISBN 0-201-14192-2.

In French :

- G. Gardarin « Bases de Données » Eyrolles
  Version gratuite sur georges.gardarin.free.fr
- **S. Miranda « *L'Art des Bases de données* » (3 Tomes), EYROLLES**
- **S. Miranda « *Bases de données : Architectures, modèles relationnels et objets, SQL3 et ODMG* », DUNOD, 2002**

- **Practicing on SQL : http://www.SQL-EX.ru**



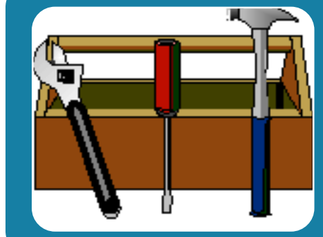
Short seminar on DATA WAREHOUSE and OLAP

DATUM ACADEMY

« **Everything (in decision support systems) is MULTI-DIMENSIONAL** »

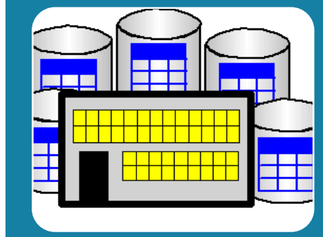
Seminar Conclusion : from DATAWARE HOUSE to …

DATUM ACADEMY



DATA PUMPING

- ETL (Extract- Transform-Load)
- COPY (Data Replication)



DATA WAREHOUSE (real built-in DB)

- METADATA & DATA REFRESH (DAY, WEEK, MONTH)
- READ-ONLY (NF2) & DECISION SUPPORT

DATA MINING and REPORTING

- Discovery mode (ML, DL)
- Verification mode (X-OLAP) ; Predefined analysis dimensions (OLAP, ..)

eMBDS

…Datalake
(for BIG DATA systems with NO SQL)

DATUM ACADEMY

DATA VIRTUALIZATION (Polystore)

- *due to Volume*
- Data ACCESS thru a common API and global data model

DATA LAKE
(virtual DB for decision support)

- Big SQL for Polystore access (SQL & NO SQL)
- Undefined analysis dimensions

MACHINE LEARNING
(Deep Learning)

- DATA MINING

eMBDS

COURSE CERTIFICATION on

https://www.DATUMACADEMY.COM