Data Basics

SQL, *intergalactic dataspeak*

Vincent Lafage

IJCLab, CNRS/IN2P3 & Université Paris-Saclay, Orsay, France

April, 15th 2026

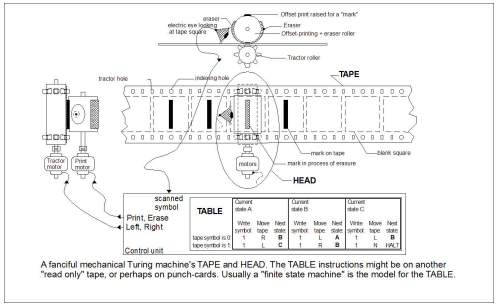
Data Bases

data (measurements?) $\Rightarrow$ informations $\Rightarrow$ knowledge

Files?

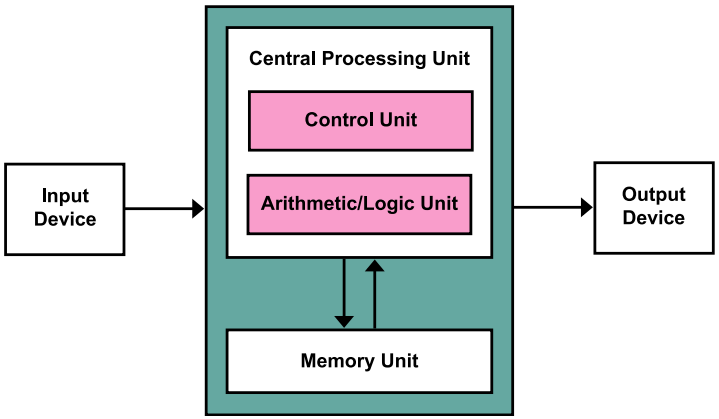
- structured metadata, but limited in number and scalability
  extensive data but no guaranteed structure
- formats...costly and frustrating to parse
- sharing...coarse granularity of rights
- access...sequential

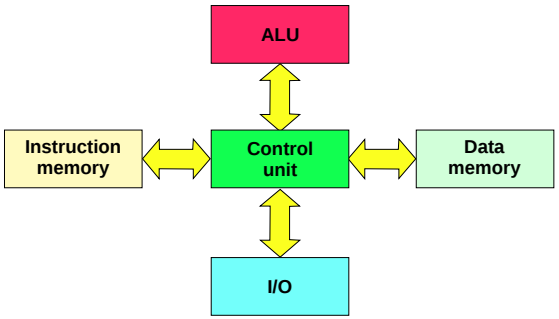
Data & processing



- Turing machine
- von Neumann architecture (Princeton architecture)

  $\Rightarrow$ von Neumann bottleneck
- Harvard architecture





- solutions...
  - ... technicaly: pipeline, cache, vectorisation
  - ... philosophicaly: functional paradigm

Abstraction

- Fortran — *expressions abstraction*

  John W. Backus 1954–1957 (1977 ACM Turing Award)

- « structured programming », — *control flow abstraction*

  ALGOL-60 Edsger Dijkstra (1972 ACM Turing Award)

  "Go To Statement Considered Harmful" 1968 doi:10.1145/362929.362947

- « object oriented » — *data abstraction*

  Smalltalk 1971, inspiré par Simula 1967, qui n'est pas objet

- « functional programming » — *iterations abstraction*

  LISP John McCarthy 1958–1960 (1971 ACM Turing Award)

  APL Kenneth E. Iverson 1957–1963 (1979 ACM Turing Award)

Data structures

data collections

(primitive types: integer, boolean, float, datetime, character… bignum, string, CLOB, BLOB)

- homogeneous: arrays `array`
- heterogeneous: `struct` / `record`
- ⇒ **table** : array of struct

  (logically equivalent to its transpose, a struct of array)
- pile, files/queues, tree, map, set, graphs…
  ⇒ indirection / reference:
  pointer for data in memory
  a blessing for structure, a curse for performance

  …but how to ensure their **persistence**?
  …but how to avoid *dangling pointers*?
  ⇒ **referential integrity**

Information

logical proposition (set-theoretic point of view)

$$\boxed{\textit{Socrates is mortal}}$$

- Syntactically *subject* "Socrates", *predicate* "is mortal".
- Semantically *theme* "Socrates", *rheme* "is mortal".

theme/topic already known from context as the sentence starts,
$\Rightarrow$ **identified**

rheme/focus the new information brought by the proposition
expresses as a **relation** between an **identifier** and **attributes** (predicate = copula + attribute)

Each sentence is a predicate of the theme : the predicate can be seen as belonging to a set, and we are increasingly specifying by making the same entity belong to the intersection of several sets.

Aristotelian definitions are this kind of intersection

$$\boxed{\text{definition} = \text{kind} + \text{specific difference}}$$

Information

Language that can reproduce this **set logic** to encode information.

relation between an **identifier** and **attributes**

$\Rightarrow$ encoded in **tables**
relational database, RDB

From this point of view, the term "object-oriented programming" is misleading, since we're not talking of "element theory", but rather of "set theory". It would be more accurate to speak of "class-oriented programming".

Database: DB

“any collection of related data”

- phone diary
- shopping list
- your banking transactions
- your five best friends
- all Facebook subscribers

storage

- On paper
- In your head
- On a computer

Create Retrieve Update Delete

are the base methods of a

DataBase Management System

DB: 2 types?

Data organisation:

...relational
one or many tables

- each table has lines and columns
- one unique key identifies each row

...non-relational
tableless

- key-value
- document: `JSON` / `XML`
- graphs

Table example

DB_CITY

Name	Country	Continent	Elevation	Latitude	Longitude	Temp.	Precip.	Rainy D.	D. Length
Dieppe	France	Europe	34	49,923195	1,076794	10,9	798	118	12,9
Lyon	France	Europe	200	45,758156	4,832499	11	770	116	12,8
Verdun	France	Europe	262	49,160833	5,388333	10,3	758	167	12,8
Annecy	France	Europe	447	45,916111	6,133056	9	1253	125	12,8
Abingdon	England	Europe	57	51,678205	-1,286612	10,2		112	12,9
Ярославль	Russia	Europe	100	57,618762	39,882289	3,9	654	223	13,2
Lund	Sweden	Europe	73	55,705051	13,191043	7	630		13,0
Cagliari	Italy	Europe	1	39,225423	9,116768	16,5	426	62	12,7
つくば	Japan	Asia	32	36,083117	140,111681	12	1490	88	12,6
東京	Japan	Asia	7	35,680252	139,771943	16,3	1520	184	12,6
Ulaanbaatar	Mongolia	Asia	1315	47,899451	106,908044	0,8	370	57	12,8
Beijing	China	Asia	54	39,909168	116,397475	12	630	75	12,7
Dieppe	Canada	America		46,094544	-64,746472	5,1	849		

...

tables are **relations**.

in the mathematic sense

hence the name “relational data base”

Relational

E.F. Codd, (1981 ACM Turing Award) *"Relational Database: A Practical Foundation for Productivity"* 1982 doi:10.1145/358396.358400

"A Relational Model of Data for Large Shared Data Banks" 1970 doi:10.1145/362384.362685

"Further Normalization of the Data Base Relational Model" *"Normalized Data Base Structure: A Brief Tutorial"* 1971 doi:10.1145/1734714.1734716

"A Relational Model of Data for Large Shared Data Banks" 1983 doi:10.1145/357980.358007

Queries: retrieving...

How many lines in the table?

```
SELECT count (*) as Nb FROM DB_CITY;
```

One table, one result corresponding to a single column of a given line:

```
SELECT Temperature FROM DB_CITY WHERE Name = `Lyon';
```

One table, one result corresponding to a single column aggregating all lines:

```
SELECT avg (Temperature) FROM DB_CITY;
```

Query: fetch...

One table, one result corresponding to many columns of a given line:

```
SELECT Temperature, Elevation FROM DB_CITY WHERE Name = `Lyon';
```

One table, one result corresponding to many columns aggregating all lines:

```
SELECT avg (Temperature) as avg_temp, Sum (Precipitation) FROM DB_CITY;
```

One table, all lines corresponding to a single column:

```
SELECT Name FROM DB_CITY;
```

One table, all lines corresponding to many columns:

```
SELECT Name, Temperature, Elevation FROM DB_CITY;
```

...sorted and / or grouped:

```
SELECT Name, Temperature, Elevation FROM DB_CITY order by temperature;
SELECT country, avg (Temperature) FROM DB_CITY group by 1;
SELECT country, avg (Temperature) FROM DB_CITY group by 1 order by 1;
SELECT country, avg (Temperature) FROM DB_CITY group by 1 order by 2;
```

Queries: towards subqueries

```
SELECT name FROM DB_CITY group by 1 having count (name)>1;

  name
--------
 Dieppe
(1 ligne)
```

Queries: subqueries

```
{SELECT * FROM DB_CITY a WHERE 1 <
    (SELECT COUNT (*) as nb FROM DB_CITY b WHERE a.name=b.name);}
```

```
  name   | country | continent | elevation | latitude  | longitude  | temperature | …
---------+---------+-----------+-----------+-----------+------------+-------------+ …
 Dieppe | France  | Europe    |        34 | 49.923195 |   1.076794 |        10.9 | …
 Dieppe | Canada  | America   |           | 46.094544 | -64.746472 |         5.1 | …
(2 lignes)
```

⇒ Beyond n-tuples...

Grammar: complex sentences with subordinate clauses

SQL : *intergalactic dataspeak*

We want the elements that are at the intersection of several sets,
and we don't care how the machine finds them as long as it's *right* and *fast*...
declarative language : as opposed to *imperative language*

1. describes *what computation* must be done and *not how* to do it
2. lacks side effects
   (or more specifically, "is referentially transparent")
3. clear correspondence to mathematical logic

cf `Prolog`

If all the functions involved in the expression are pure functions (deterministic + with no edge effects), then the expression is referentially transparent.

A referentially transparent expression may involve impure functions and therefore edge effects. Referential transparency is the cornerstone of functional programming.

1. **DDL** Data Definition Language — `CREATE`, `ALTER`, `DROP`
2. **DML** Data Manipulation Language — `INSERT`, `UPDATE`, `DELETE`
3. **DQL** Data Query Language — `SELECT`
   - information about an instance of an entity
   - several pieces of information about an instance of an entity
   - one piece of information about several instances of an entity
     $\Rightarrow$ aggregate: sum, product, average and other statistics (variance, min, max, median...)
   - several pieces of information on several instances of an entity
     $\Rightarrow$ *cursor* : result of a query $\sim$ structure of a table
4. **DCL** Data Control Language — `GRANT`, `REVOKE`
5. **TCL** Transaction Control Language — `COMMIT`, `ROLLBACK`

DDL

```sql
drop table DB_CITY;
create table DB_CITY (
    Name            varchar (40) NOT NULL,
    Country         varchar (40) NOT NULL,
    Continent       varchar (40) NOT NULL,
    Elevation       integer,
    Latitude        double precision,
    Longitude       double precision,
    Temperature     float,
    Precipitation   integer,
    Rainy_Day       integer,
    Day_Length      numeric (3,1)
);

comment on table DB_CITY is 'Cities with their some of their physical geography informations';

comment on column DB_CITY.Name          is 'Name, either in the original script or romanized English';
comment on column DB_CITY.Country       is 'Country, with romanized English name';
comment on column DB_CITY.Continent     is 'Continent, with romanized English name';
comment on column DB_CITY.Elevation     is 'Average elevation above sea level, in meter';
comment on column DB_CITY.Latitude      is 'Latitude, in degree';
comment on column DB_CITY.Longitude     is 'Longitude, in degree';
comment on column DB_CITY.Temperature   is 'Yearly Average Temperature, in Celsius degree';
comment on column DB_CITY.Precipitation is 'Yearly cumulative Precipitation, in millimeter';
comment on column DB_CITY.Rainy_Day     is 'Number of Rainy_Day, per year';
comment on column DB_CITY.Day_Length    is 'Yearly Average Day_Length, in hours';
```

Mass loading

```
COPY DB_CITY (Name, Country, Continent, Elevation, Latitude, Longitude, Temperature,
              Precipitation, Rainy_Day, Day_Length) FROM stdin (DELIMITER '|');
Dieppe|France|Europe|34|49.923195|1.076794|10.9|798|118|12.9
Lyon|France|Europe|200|45.758156|4.832499|11|770|116|12.8
Verdun|France|Europe|262|49.160833|5.388333|10.3|758|167|12.8
Annecy|France|Europe|447|45.916111|6.133056|9|1253|125|12.8
Abingdon|England|Europe|57|51.678205|-1.286612|10.2|\N|112|12.9
Ярославль|Russia|Europe|100|57.618762|39.882289|3.9|654|223|13.2
Lund|Sweden|Europe|73|55.705051|13.191043|7|630|\N|13.0
Cagliari|Italy|Europe|1|39.225423|9.116768|16.5|426|62|12.7
□□□|Japan|Asia|32|36.083117|140.111681|12|1490|88|12.6
□□|Japan|Asia|7|35.680252|139.771943|16.3|1520|184|12.6
Ulaanbaatar|Mongolia|Asia|1315|47.899451|106.908044|0.8|370|57|12.8
Beijing|China|Asia|54|39.909168|116.397475|12|630|75|12.7
Dieppe|Canada|America|\N|46.094544|-64.746472|5.1|849|\N|\N
\.
```

RBD lexicon

- table (entity)
- field
- identifier
- sequence
- query
- view
- index
- jointure
- constraint
  - mandatory fields or “NOT NULL”
  - unique value
  - referential integrity
- trigger
- stored procedure

a good identifier

entity = class: a set of objects of the same type

$\Rightarrow$ arranged in a table

- Objects make up the table rows...
- ... whose columns form the structure

This structure can refer to other structures stored in other tables.
To facilitate this reference, we can **identify** the members of the set.
For example, with the line number of the object in the table.
Many DBMSs have a default identifier (Object ID) for everything they manage.
Best practice, however, is to manage this ID explicitly as the first column of each table.

a good identifier

good identifier $\Leftrightarrow$ as meaningless as possible

... real serial number, matricule number

Not the french social security number: for this number, each field has a meaning (`G-YY-MM-PP-AAA-NNN-##`). This is not flexible enough if the system needs to evolve. For example,

- Management of centuries is flawed
- it doesn't trivially extend to Europe.

All that's required of an identifier is that it be

1. present in each row (mandatory, not NULL)
2. unique in the column (otherwise, it won't unequivocally identify the row)

Identifiers are not required to be *neither contiguous, nor increasing* **integers** are the canonical identifier par excellence: compact, concise, *CPU bread&butter*

By the way, in most object oriented designs, it's all too easy to forget that each object has an identifier: its memory storage address (an integer...)

$\Rightarrow$ sequence

Normalisation

"Process of organizing the columns (attributes) and tables (relationships) of a relational database to reduce data redundancy and improve data integrity".
"Free the database from anomalies during modifications".
E.F. Codd, *"Further Normalization of the Data Base Relational Model"*

- prerequisite: the values of the various attributes are functionally dependent on the primary key
- 1NF: allow querying and manipulation of data using a universal data sub-language based on first-order logic. permettre le requêtage et la manipulation des données en utilisant un "universal data sub-language" fondé sur la logique du premier ordre.
  $\Rightarrow$ one field per column and only one
  1. all data is atomic;
  2. contain a scalar value
  3. contain non-repeating values (the opposite case is to put a list in a single attribute);
  4. are constant over time (use date of birth rather than age, for example).

Normalization

DB_PUBLI

id_article	nom_auteur	prenom_auteur	email_auteur	date_article	titre_article
1	Dupont	Jacques	jacques_dupont@email.com	26/01/2010	Wall Street repasse dans le vert
2	Dubois	Marcel	mdubois@email.com	27/01/2010	La Fed garde son taux directeur inchangé
3	Dupont	Jacques	jacques_dupont@email.com	28/01/2010	Des pensionnés très occupés!
4	Leroy	Nicole	nicole.leroy@email.com	28/01/2010	L'entretien du réseau routier laisse à désirer

we'd like to rationalize this: information factorization

DB_ARTICLE

id_arti	id_aute	date_arti	titre_arti
1	101	26/01/2010	Wall Street repasse dans le vert
2	102	27/01/2010	La Fed garde son taux directeur inchangé
3	101	28/01/2010	Des pensionnés très occupés!
4	103	28/01/2010	L'entretien du réseau routier laisse à désirer

DB_AUTEUR

id_aute	nom_aute	prenom_aute	email_aute
101	Dupont	Jacques	jacques_dupont@email.com
102	Dubois	Marcel	mdubois@email.com
103	Leroy	Nicole	nicole.leroy@email.com

Join

The real Power of databases!

```
SELECT * FROM DB_PUBLI;
```

$\Leftrightarrow$

```
SELECT * FROM DB_ARTICLE JOIN DB_AUTEUR on id_aut;
```

$\Leftrightarrow$

```
SELECT * FROM DB_ARTICLE NATURAL JOIN DB_AUTEUR;
```

$\Leftrightarrow$

```
SELECT * FROM DB_ARTICLE, DB_AUTEUR WHERE DB_ARTICLE.id_aut = DB_AUTEUR.id_aut;
```

Normal forms

1NF, 2NF, 3NF, BCNF (Boyce Codd), 4NF, 5NF, DKNF (Domain-key), 6NF

1. free the relationship collection from unwanted dependencies on insertion, update and deletion
2. reduce the need to restructure the relationship collection when new data types are introduced, thus increasing the application's life expectancy;
3. make the relational model more informative for users

- limit data redundancy (multiple writes);
- limit data inconsistencies that could render them unusable (multiple writes);
- avoid updating processes (rewrites).

1FN = The key

2FN = All the key

Memory/time trade-off

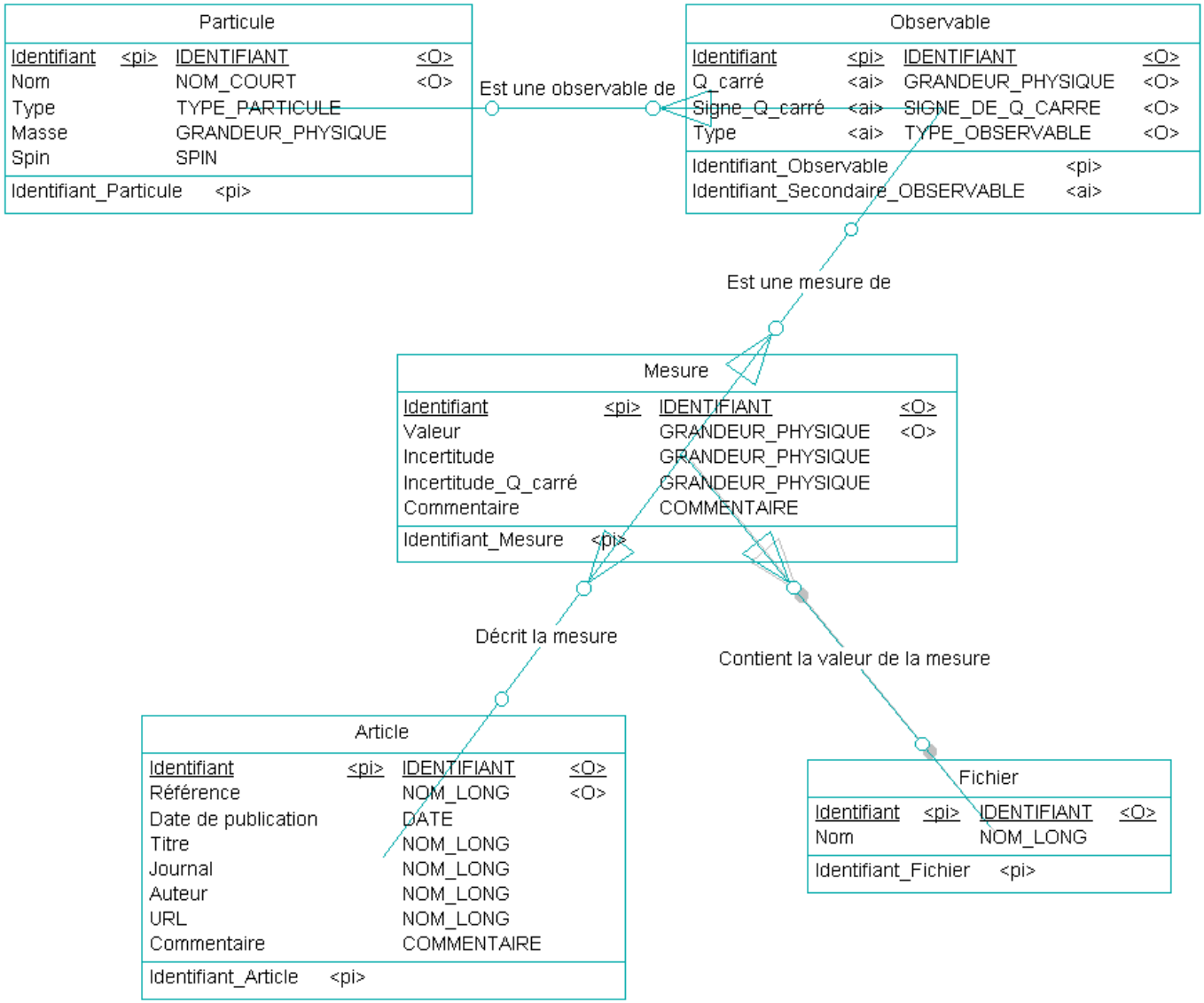
Read/write trade-off

3FN = Only the key.

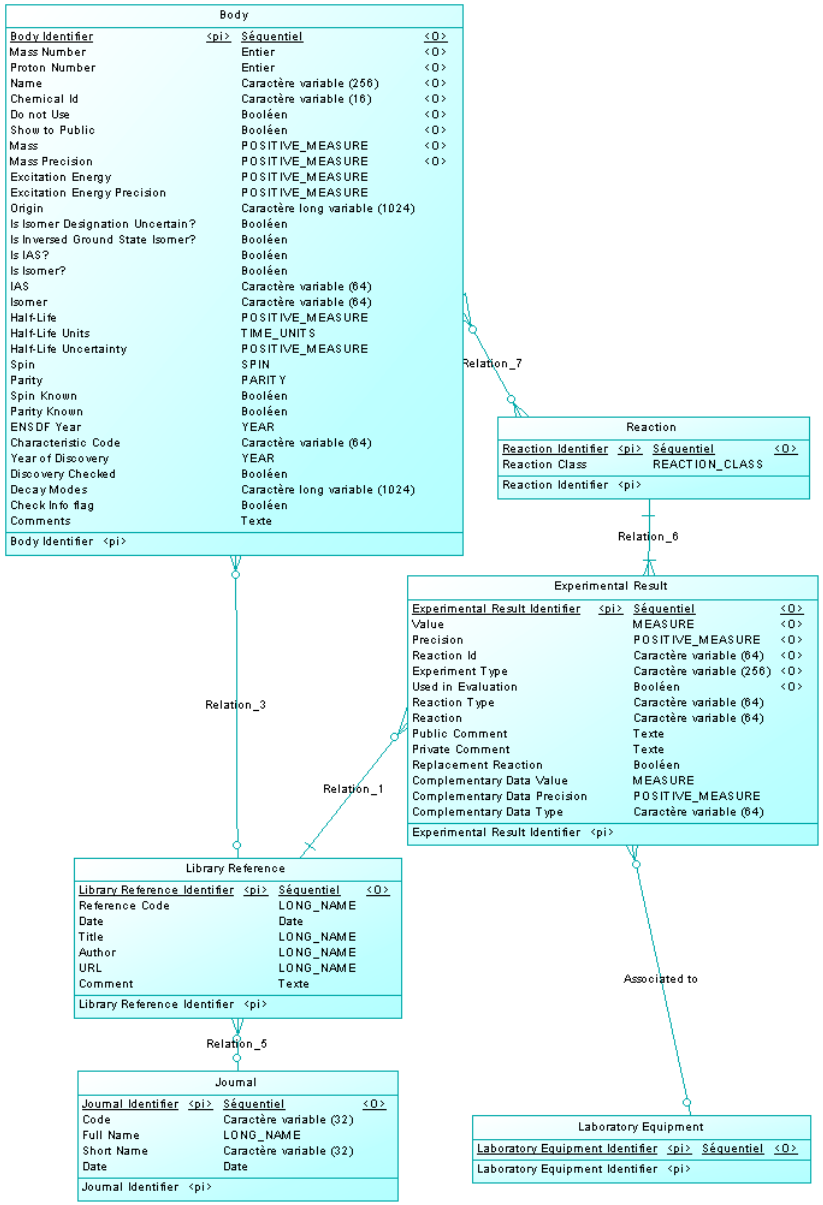
Data modeling

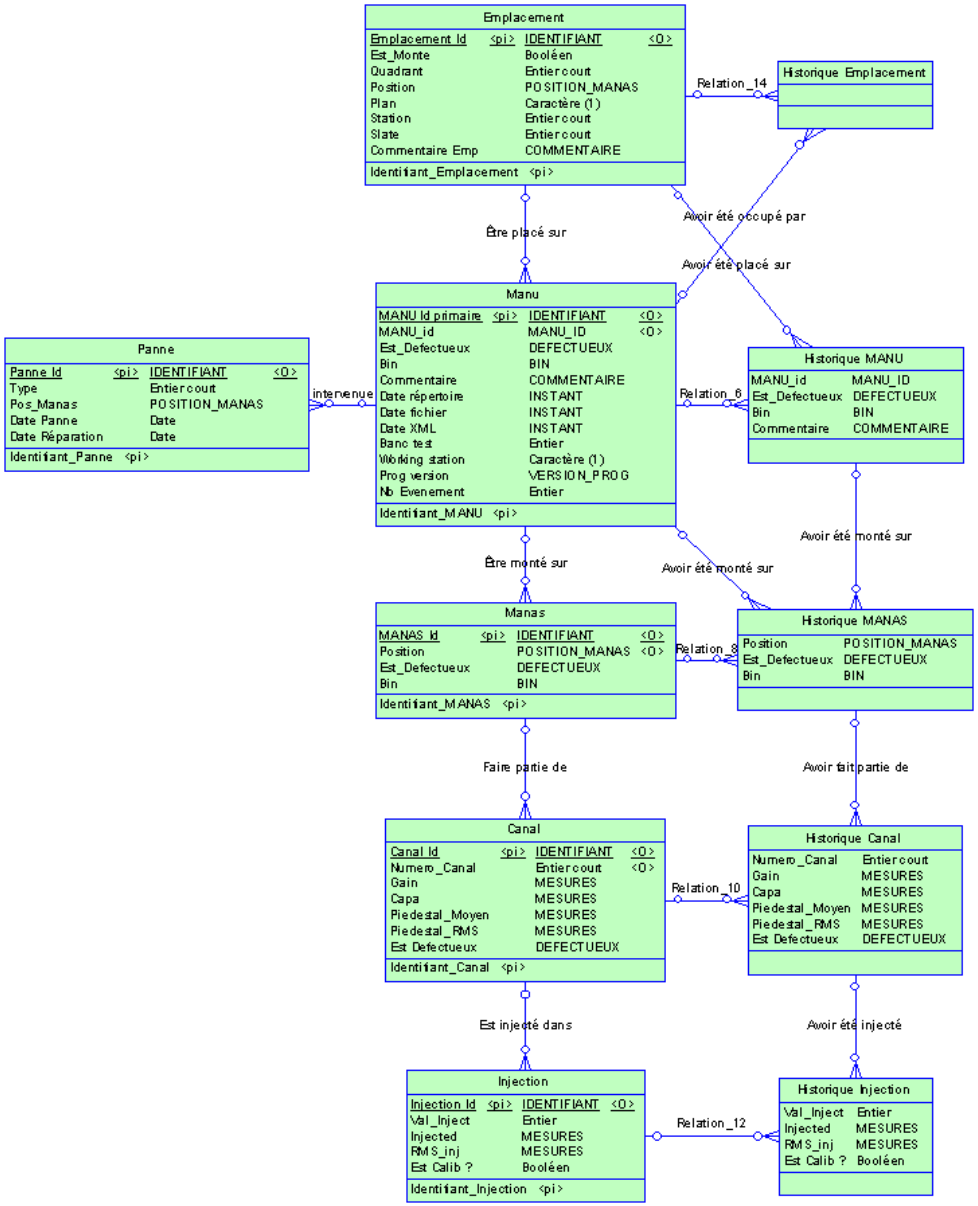
Merise method, “Entity–relationship” (entity-association)
MCD-MLD-MPD: conceptual-logical-physical models

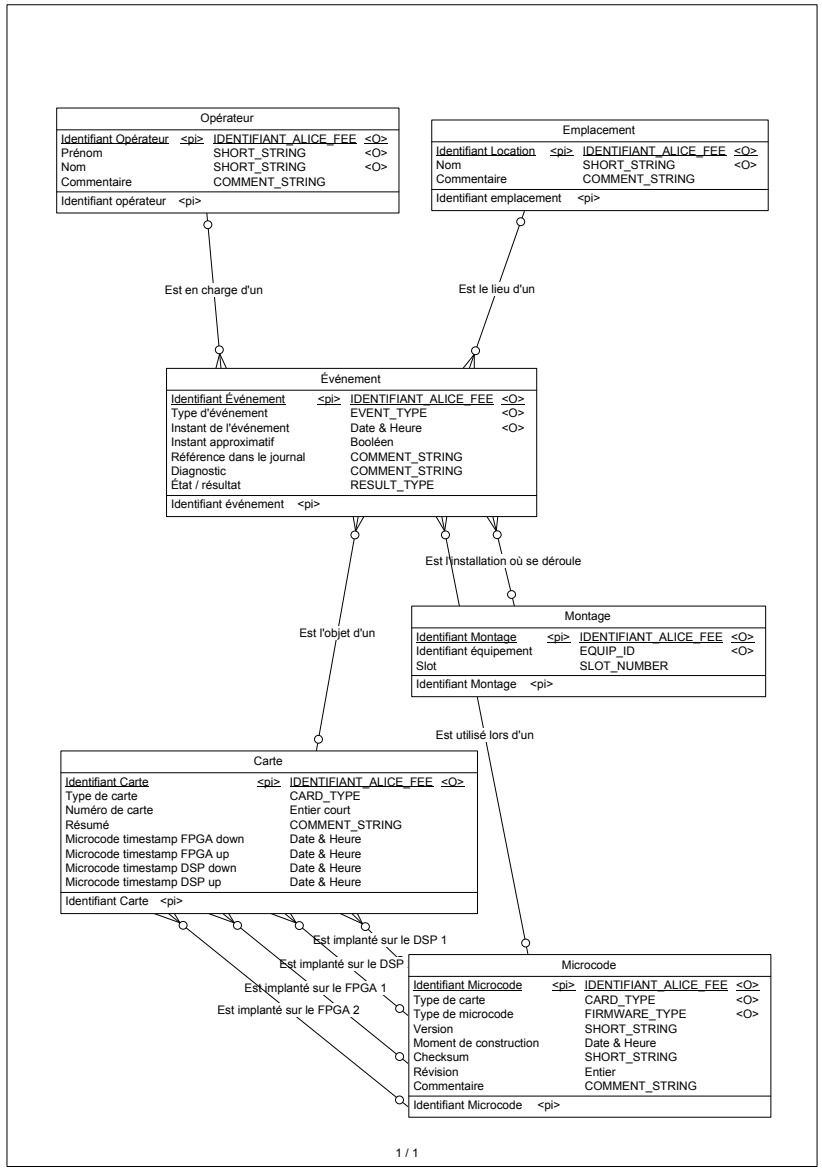
Familiarity with object methods such as UML


Particule
Identifiant <pi> IDENTIFIANT <O>
Nom NOM_COURT <O>
Type TYPE_PARTICULE
Masse GRANDEUR_PHYSIQUE
Spin SPIN
Identifiant_Particule <pi>
Est une observable de
Observable
Identifiant <pi> IDENTIFIANT <O>
Q_carré <ai> GRANDEUR_PHYSIQUE <O>
Signe_Q_carré <ai> SIGNE_DE_Q_CARRE <O>
Type <ai> TYPE_OBSERVABLE <O>
Identifiant_Observable <pi>
Identifiant_Secondaire_OBSERVABLE <ai>
Est une mesure de
Mesure
Identifiant <pi> IDENTIFIANT <O>
Valeur GRANDEUR_PHYSIQUE <O>
Incertitude GRANDEUR_PHYSIQUE
Incertitude_Q_carré GRANDEUR_PHYSIQUE
Commentaire COMMENTAIRE
Identifiant_Mesure <pi>
Décrit la mesure
Contient la valeur de la mesure
Article
Identifiant <pi> IDENTIFIANT <O>
Référence NOM_LONG <O>
Date de publication DATE
Titre NOM_LONG
Journal NOM_LONG
Auteur NOM_LONG
URL NOM_LONG
Commentaire COMMENTAIRE
Identifiant_Article <pi>
Fichier
Identifiant <pi> IDENTIFIANT <O>
Nom NOM_LONG
Identifiant_Fichier <pi>

Nuclei


Body
Body Identifier <pi> Séquentiel <O>
Mass Number Entier <O>
Proton Number Entier <O>
Name Caractère variable (256) <O>
Chemical Id Caractère variable (16) <O>
Do not Use Booléen <O>
Show to Public Booléen <O>
Mass POSITIVE_MEASURE <O>
Mass Precision POSITIVE_MEASURE <O>
Excitation Energy POSITIVE_MEASURE
Excitation Energy Precision POSITIVE_MEASURE
Origin Caractère long variable (1024)
Is Isomer Designation Uncertain? Booléen
Is Inversed Ground State Isomer? Booléen
Is IAS? Booléen
Is Isomer? Booléen
IAS Caractère variable (64)
Isomer Caractère variable (64)
Half-Life POSITIVE_MEASURE
Half-Life Units TIME_UNITS
Half-Life Uncertainty POSITIVE_MEASURE
Spin SPIN
Parity PARITY
Spin Known Booléen
Parity Known Booléen
ENSDF Year YEAR
Characteristic Code Caractère variable (64)
Year of Discovery YEAR
Discovery Checked Booléen
Decay Modes Caractère long variable (1024)
Check Info flag Booléen
Comments Texte
Body Identifier <pi>
Relation_7
Reaction
Reaction Identifier <pi> Séquentiel <O>
Reaction Class REACTION_CLASS
Reaction Identifier <pi>
Relation_6
Experimental Result
Experimental Result Identifier <pi> Séquentiel <O>
Value MEASURE <O>
Precision POSITIVE_MEASURE <O>
Reaction Id Caractère variable (64) <O>
Experiment Type Caractère variable (256) <O>
Used in Evaluation Booléen <O>
Reaction Type Caractère variable (64)
Reaction Caractère variable (64)
Public Comment Texte
Private Comment Texte
Replacement Reaction Booléen
Complementary Data Value MEASURE
Complementary Data Precision POSITIVE_MEASURE
Complementary Data Type Caractère variable (64)
Experimental Result Identifier <pi>
Relation_3
Relation_1
Library Reference
Library Reference Identifier <pi> Séquentiel <O>
Reference Code LONG_NAME
Date Date
Title LONG_NAME
Author LONG_NAME
URL LONG_NAME
Comment Texte
Library Reference Identifier <pi>
Relation_5
Journal
Journal Identifier <pi> Séquentiel <O>
Code Caractère variable (32)
Full Name LONG_NAME
Short Name Caractère variable (32)
Date Date
Journal Identifier <pi>
Associated to
Laboratory Equipment
Laboratory Equipment Identifier <pi> Séquentiel <O>
Laboratory Equipment Identifier <pi>


Emplacement
Emplacement Id <pi> IDENTIFIANT <O>
Est_Monte Booléen
Quadrant Entier court
Position POSITION_MANAS
Plan Caractère (1)
Station Entier court
Slate Entier court
Commentaire Emp COMMENTAIRE
Identifiant_Emplacement <pi>
Relation_14
Historique Emplacement
Être placé sur
Avoir été occupé par
Avoir été placé sur
Manu
MANU Id primaire <pi> IDENTIFIANT <O>
MANU_id MANU_ID <O>
Est_Defectueux DEFECTUEUX
Bin BIN
Commentaire COMMENTAIRE
Date répertoire INSTANT
Date fichier INSTANT
Date XML INSTANT
Banc test Entier
Working station Caractère (1)
Prog version VERSION_PROG
Nb Evenement Entier
Identifiant_MANU <pi>
Panne
Panne Id <pi> IDENTIFIANT <O>
Type Entier court
Pos_Manas POSITION_MANAS
Date Panne Date
Date Réparation Date
Identifiant_Panne <pi>
intervenue
Relation_6
Historique MANU
MANU_id MANU_ID
Est_Defectueux DEFECTUEUX
Bin BIN
Commentaire COMMENTAIRE
Être monté sur
Avoir été monté sur
Avoir été monté sur
Manas
MANAS Id <pi> IDENTIFIANT <O>
Position POSITION_MANAS <O>
Est_Defectueux DEFECTUEUX
Bin BIN
Identifiant_MANAS <pi>
Relation_8
Historique MANAS
Position POSITION_MANAS
Est_Defectueux DEFECTUEUX
Bin BIN
Faire partie de
Avoir fait partie de
Canal
Canal Id <pi> IDENTIFIANT <O>
Numero_Canal Entier court <O>
Gain MESURES
Capa MESURES
Piedestal_Moyen MESURES
Piedestal_RMS MESURES
Est Defectueux DEFECTUEUX
Identifiant_Canal <pi>
Relation_10
Historique Canal
Numero_Canal Entier court
Gain MESURES
Capa MESURES
Piedestal_Moyen MESURES
Piedestal_RMS MESURES
Est Defectueux DEFECTUEUX
Est injecté dans
Avoir été injecté
Injection
Injection Id <pi> IDENTIFIANT <O>
Val_Inject Entier
Injected MESURES
RMS_inj MESURES
Est Calib ? Booléen
Identifiant_Injection <pi>
Relation_12
Historique Injection
Val_Inject Entier
Injected MESURES
RMS_inj MESURES
Est Calib ? Booléen


Opérateur
Identifiant Opérateur <pi> IDENTIFIANT_ALICE_FEE <O>
Prénom SHORT_STRING <O>
Nom SHORT_STRING <O>
Commentaire COMMENT_STRING
Identifiant opérateur <pi>
Emplacement
Identifiant Location <pi> IDENTIFIANT_ALICE_FEE <O>
Nom SHORT_STRING <O>
Commentaire COMMENT_STRING
Identifiant emplacement <pi>
Est en charge d'un
Est le lieu d'un
Événement
Identifiant Événement <pi> IDENTIFIANT_ALICE_FEE <O>
Type d'événement EVENT_TYPE <O>
Instant de l'événement Date & Heure <O>
Instant approximatif Booléen
Référence dans le journal COMMENT_STRING
Diagnostic COMMENT_STRING
État / résultat RESULT_TYPE
Identifiant événement <pi>
Est l'installation où se déroule
Est l'objet d'un
Montage
Identifiant Montage <pi> IDENTIFIANT_ALICE_FEE <O>
Identifiant équipement EQUIP_ID <O>
Slot SLOT_NUMBER
Identifiant Montage <pi>
Est utilisé lors d'un
Carte
Identifiant Carte <pi> IDENTIFIANT_ALICE_FEE <O>
Type de carte CARD_TYPE
Numéro de carte Entier court
Résumé COMMENT_STRING
Microcode timestamp FPGA down Date & Heure
Microcode timestamp FPGA up Date & Heure
Microcode timestamp DSP down Date & Heure
Microcode timestamp DSP up Date & Heure
Identifiant Carte <pi>
Est implanté sur le DSP 1
Est implanté sur le DSP
Est implanté sur le FPGA 1
Est implanté sur le FPGA 2
Microcode
Identifiant Microcode <pi> IDENTIFIANT_ALICE_FEE <O>
Type de carte CARD_TYPE <O>
Type de microcode FIRMWARE_TYPE <O>
Version SHORT_STRING
Moment de construction Date & Heure
Checksum SHORT_STRING
Révision Entier
Commentaire COMMENT_STRING
Identifiant Microcode <pi>
1 / 1

Constraint: CREATE TABLE

```
CREATE TABLE `full_att` (
  `time` datetime(3),
  `attr_id` INT2 UNSIGNED,
  `value` double DEFAULT NULL
);
```

```
CREATE TABLE `full_att` (
  `time` datetime(3),
  `attr_id` INT2 UNSIGNED,
  `value` double
);
```

Constraint: NOT NULL

```
CREATE TABLE products (
  product_no integer NOT NULL,
  name text NOT NULL,
  price numeric
);
```

Constraint: CHECK

```
CREATE TABLE products (
  product_no integer,
  name text,
  price numeric CHECK (price > 0)
);
```

```
CREATE TABLE products (
  product_no integer,
  name text,
  price numeric CONSTRAINT positive_price CHECK (price > 0)
);
```

```
CREATE TABLE products (
  product_no integer,
  name text,
  price numeric CHECK (price > 0),
  discounted_price numeric CHECK (discounted_price > 0),
  CHECK (price > discounted_price)
);
```

Constraint: UNIQUE

```sql
CREATE TABLE products (
  product_no integer UNIQUE,
  name text,
  price numeric
);
```

```sql
CREATE TABLE products (
  product_no integer,
  name text,
  price numeric,
  UNIQUE (product_no)
);
```

```sql
CREATE TABLE example (
  a integer,
  b integer,
  c integer,
  UNIQUE (a, c)
);
```

Constraint: PRIMARY KEY

```
CREATE TABLE products (
  product_no integer UNIQUE NOT NULL,
  name text,
  price numeric
);
```

```
CREATE TABLE products (
  product_no integer PRIMARY KEY,
  name text,
  price numeric
);
```

```
CREATE TABLE example (
  a integer,
  b integer,
  c integer,
  PRIMARY KEY (a, c)
);
```

Constraint: FOREIGN KEY

Referential integrity

```
CREATE TABLE products (
  product_no integer PRIMARY KEY,
  name text,
  price numeric
);
```

```
CREATE TABLE orders (
  order_id integer PRIMARY KEY,
  product_no integer REFERENCES products (product_no),
  quantity integer
);
```

```
CREATE TABLE orders (
  order_id integer PRIMARY KEY,
  product_no integer REFERENCES products,
  quantity integer
);
```

```
CREATE TABLE t1 (
  a integer PRIMARY KEY,
  b integer,
  c integer,
  FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)
);
```

Constraint: Self Reference / Tree

```
CREATE TABLE tree (
  node_id integer PRIMARY KEY,
  parent_id integer REFERENCES tree,
  name text,
  ...
);
```

Constraint: Multiple References

```
CREATE TABLE products (
  product_no integer PRIMARY KEY,
  name text,
  price numeric
);

CREATE TABLE orders (
  order_id integer PRIMARY KEY,
  shipping_address text,
  ...
);

CREATE TABLE order_items (
  product_no integer REFERENCES products,
  order_id integer REFERENCES orders,
  quantity integer,
  PRIMARY KEY (product_no, order_id)
);
```

Constraint: CASCADE

```
CREATE TABLE products (
  product_no integer PRIMARY KEY,
  name text,
  price numeric
);

CREATE TABLE orders (
  order_id integer PRIMARY KEY,
  shipping_address text,
  ...
);

CREATE TABLE order_items (
  product_no integer REFERENCES products ON DELETE RESTRICT,
  order_id integer REFERENCES orders ON DELETE CASCADE,
  quantity integer,
  PRIMARY KEY (product_no, order_id)
);
```

Constraint: CREATE INDEX

```
CREATE TABLE `full_att` (
  `time` datetime(3) NOT NULL,
  `attr_id` INT2 UNSIGNED NOT NULL,
  `value` double DEFAULT NULL
);
```

The engine behind contemporary Web

```
CREATE OR REPLACE INDEX idx_time_full ON full_att (time);
CREATE OR REPLACE INDEX idx_attr_full ON full_att (attr_id);
CREATE OR REPLACE INDEX idx_time_attr_full ON full_att (time, attr_id);
```

Constraint: ALTER TABLE

```
CREATE TABLE `adt` (
`ID` smallint(5) unsigned zerofill NOT NULL AUTO_INCREMENT,
`time` datetime DEFAULT NULL,
`full_name` varchar(191) NOT NULL DEFAULT '',
`device` varchar(150) NOT NULL DEFAULT '',
...
`substitute` smallint(9) NOT NULL DEFAULT '0',
PRIMARY KEY (`ID`,`full_name`),
UNIQUE KEY `ID_2` (`ID`),
UNIQUE KEY `full_name` (`full_name`),
KEY `ID` (`ID`)
) COMMENT='Attribute Definition Table';
```

```
ALTER TABLE `adt` DROP PRIMARY KEY;
ALTER TABLE `adt` ADD CONSTRAINT pk_adt PRIMARY KEY (`ID`);
CREATE OR REPLACE INDEX idx_adt_full_name ON adt (full_name);
```

CREATE TRIGGER

```
CREATE TRIGGER duplicate_att_00001
   AFTER INSERT ON att_00001
   FOR EACH ROW
   INSERT INTO full_att (time, attr_id, value) VALUES(NEW.time, 1, NEW.value);
```

```
CREATE TRIGGER check_update
   BEFORE UPDATE ON accounts
   FOR EACH ROW
   EXECUTE FUNCTION check_account_update();
```

```
CREATE TRIGGER check_update
   BEFORE UPDATE ON accounts
   FOR EACH ROW
   WHEN (OLD.balance IS DISTINCT FROM NEW.balance)
   EXECUTE FUNCTION check_account_update();
```

Data-processing interaction

thick or thin mapping issues
flexibility & security in modeling:

- views / stored query
- stored procedures

Declarative languages are powerful, but they can't do everything, especially when studying the interactions of objects of the same kind, they can be grouped (`GROUP BY`), you can sort them (`ORDER BY`) but how can we study the variations between one object and the next in the query?
cf temporal database

Most data processing are written using **imperative** languages: reading files, data acquisition, text parsing, optimisation method, graphical user interface...

How to interface these two styles of language?

How to represent `NULL` values?

- with a library
- embedded SQL

embedded SQL

```
#include <stdio.h>

EXEC SQL BEGIN DECLARE SECTION;
  int x;
  int x_indicator;
EXEC SQL END DECLARE SECTION;

int i;
void main () {
  EXEC SQL CONNECT TO database_or_server USER Josephine;
  EXEC SQL DECLARE example_cursor CURSOR FOR SELECT col_1 FROM T;
  EXEC SQL OPEN example_cursor;
  for (i=0; i<10; ++i) {
    EXEC SQL FETCH example_cursor INTO :x INDICATOR :x_indicator;
    if (x_indicator < 0) printf("?\n");
    else printf("\%d\n",x);
  }
  EXEC SQL CLOSE example_cursor;
  EXEC SQL DISCONNECT database_or_server;
}
```

embedded SQL

```c
#include <stdio.h>
EXEC SQL INCLUDE sqlca;

int main() {
    EXEC SQL CONNECT TO 'dbname' AS 'mydb' USER 'username' PASSWORD 'password';

    EXEC SQL DECLARE bin_cursor BINARY CURSOR FOR SELECT id, price FROM products;
    EXEC SQL OPEN bin_cursor;

    EXEC SQL BEGIN DECLARE SECTION;
    int id;
    float price;
    EXEC SQL END DECLARE SECTION;

    while (1) {
        EXEC SQL FETCH bin_cursor INTO :id, :price;
        if (sqlca.sqlcode != 0) break; // Fin du curseur
        printf("ID: %d, Price: %.2f\n", id, price);
    }

    EXEC SQL CLOSE bin_cursor;
    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT;
    return 0;
}
```

Breakdown of a file into a set of tables

You can send a series of requests from an application to a database and lose time in network latency.

Ideally, the query should be executed on the database server.

```
CREATE OR REPLACE FUNCTION tabsin(_p1 float, _p2 float, _p3 float)
RETURNS void AS $$
DECLARE _i float = _p1; _do float = _p2; _step float = _p3;
BEGIN
  IF NOT EXISTS(SELECT relname FROM pg_class
    WHERE relname = 'tabsin' AND relkind = 'r' AND pg_table_is_visible(oid)) THEN
    RAISE NOTICE 'Create table tabsin';
    CREATE TEMP TABLE tabsin (x NUMERIC(5,4) PRIMARY KEY, fx NUMERIC(5,4));
  ELSE
    RAISE NOTICE 'Deleting all records from table tabsin';
    TRUNCATE TABLE tabsin;
  END IF;
  WHILE _i < _do LOOP
    INSERT INTO tabsin VALUES(CAST(_i AS NUMERIC(5,4)), SIN(_i));
    _i := _i + _step;
  END LOOP;
END;
$$ LANGUAGE plpgsql;
```

```
Declare
   -- Déclaration du curseur
   CURSOR C_EMP (PC$Job IN EMP.job%Type) IS
   Select
      *
   From
      EMP
   Where
      job = PC$Job
   ;

Begin
   For Cur IN C_EMP ('SALESMAN') Loop
     dbms_output.put_line (To_char (Cur.empno) || ' - ' || Cur.ename) ;
   End loop ;
End ;
```

Text or binary cursors?

A soberly technical problem: when bringing back a cursor, if it contains numbers (integers, fixed or floating point) it may be advantageous not to transmit it in text form: more compact in binary, we avoid double conversion with the rounding errors induced.

Gain in speed (CPU), bandwidth (network) and accuracy

Query optimisation

`EXPLAIN` “table scan” ou “sequential scan” *vs.* “indexed scan”
dichotomous search (pure for unique keys) logarithmic
join
cost-based strategy (predetermined costs)
strategy following tables-statistics

EXPLAIN in practice

Simple query on a table with a single integer column and 10000 rows:

```
EXPLAIN SELECT * FROM foo;
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on foo  (cost=0.00..155.00 rows=10000 width=4)
(1 row)
```

```
EXPLAIN SELECT * FROM foo WHERE i = 4;
                         QUERY PLAN
-------------------------------------------------------------
 Index Scan using fi on foo  (cost=0.00..5.98 rows=1 width=4)
   Index Cond: (i = 4)
(2 rows)
```

EXPLAIN in practice

```
EXPLAIN SELECT sum(i) FROM foo WHERE i < 10;

                              QUERY PLAN
-----------------------------------------------------------------------
 Aggregate  (cost=23.93..23.93 rows=1 width=4)
   ->  Index Scan using fi on foo  (cost=0.00..23.92 rows=6 width=4)
         Index Cond: (i < 10)
(3 rows)
```

```
PREPARE query(int, int) AS SELECT sum(bar) FROM test
    WHERE id > $1 AND id < $2
    GROUP BY foo;

EXPLAIN ANALYZE EXECUTE query(100, 200);

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=39.53..39.53 rows=1 width=8) (actual time=0.661..0.672 rows=7 loops=1)
   ->  Index Scan using test_pkey on test  (cost=0.00..32.97 rows=1311 width=8) (actual time=0.050..0.395 row
         Index Cond: ((id > $1) AND (id < $2))
 Total runtime: 0.851 ms
(4 rows)
```

ODBC: Open DataBase Connectivity

- **Definition**: Standardized interface for accessing databases.
- **Purpose**: Enables applications (e.g., SQL clients) to communicate with various DBMS (Database Management Systems).
- **Origin**: Developed by Microsoft in the 1990s, based on the Call-Level Interface (CLI) API.

How It Works

- **Architecture**:
  - Application (e.g., SQL client).
  - `ODBC` Driver specific to the DBMS.
  - Driver Manager.
  - Target database (PostgreSQL, MySQL, etc.).
- **Key Role**: Abstracts DBMS-specific details for universal compatibility.

Advantages and Limitations

Advantages:

- Application portability.
- Flexibility (multi-DBMS support).
- Wide adoption (Windows, Linux, etc.).

Limitations:

- Dependency on drivers.
- Potentially lower performance vs. native APIs.
- Initial configuration required.

Example Use Case with SQL

- Connecting via `ODBC` in an SQL tool (e.g., DBeaver, SSMS):
  1. Set up a Data Source Name (DSN).
  2. Use a connection string (e.g., `"DRIVER=SQL Server;SERVER=localhost;DATABASE=myDB;"`).
  3. Execute standard SQL queries.
- **Conclusion**: `ODBC` simplifies data access in heterogeneous environments.

Dynamic SQL

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "INSERT INTO test1 VALUES(?, ?);";
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE mystmt FROM :stmt;
 ...
EXEC SQL EXECUTE mystmt USING 42, 'foobar';
...
EXEC SQL DEALLOCATE PREPARE mystmt;
```

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "SELECT a, b, c FROM test1 WHERE a > ?";
int v1, v2;
VARCHAR v3[50];
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE mystmt FROM :stmt;
 ...
EXEC SQL EXECUTE mystmt INTO :v1, :v2, :v3 USING 37;
```

Dynamic SQL

```
EXEC SQL BEGIN DECLARE SECTION;
char dbaname[128];
char datname[128];
char *stmt = "SELECT u.usename as dbaname, d.datname "
             "  FROM pg_database d, pg_user u "
             "  WHERE d.datdba = u.usesysid";
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO testdb AS con1 USER testuser;
EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;

EXEC SQL PREPARE stmt1 FROM :stmt;

EXEC SQL DECLARE cursor1 CURSOR FOR stmt1;
EXEC SQL OPEN cursor1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
    EXEC SQL FETCH cursor1 INTO :dbaname,:datname;
    printf("dbaname=%s, datname=%s\n", dbaname, datname);
}

EXEC SQL CLOSE cursor1;

EXEC SQL COMMIT;
EXEC SQL DISCONNECT ALL;
```

DOMAIN

```
CREATE DOMAIN us_postal_code AS TEXT
CHECK(
   VALUE ~ '^\d{5}$'
OR VALUE ~ '^\d{5}-\d{4}$'
);

CREATE TABLE us_snail_addy (
  address_id SERIAL PRIMARY KEY,
  street1 TEXT NOT NULL,
  street2 TEXT,
  street3 TEXT,
  city TEXT NOT NULL,
  postal us_postal_code NOT NULL
);
```