Date de dernière modification *: 30/6/2015*

Annexe **10**

Les bases de données non relationnelles

Outre des informations générales sur le terme NoSQL et sur la variété des SGBD, cette annexe développe de manière pratique la notion de conversion de modèles. Des données organisées selon un modèle peuvent être transformées sans perte de manière à être conformes à un autre modèles. On applique ce principe à quelques modèles dont ceux que recouvre l'appellation NoSQL.

A10.1QUE SIGNIFIE NoSQL ?

NoSQL est le #tag choisi, un peu au hasard, par un développeur londonien pour identifier une réunion informelle qu'il organisait à San Francisco en 2009. Cette réunion rassemblait divers acteurs du domaine des nouvelles technologies des bases de données, invités à présenter leurs offres. Très vite adopté pour désigner la famille des nouveaux SGBD, on lui a attribué dans un premier temps une *sémantique guerrière* de **rejet de SQL**, considéré (une fois de plus depuis 1980 !) comme dépassé. Progressivement, une nouvelle interprétation plus conviviale s'est imposée : ***Not only SQL***. Comme nous l'avons vu au chapitre 10, celle-ci n'est guère plus pertinente, mais a le mérite de mettre en évidence l'élargissement de l'écosystème des bases de données. *Source* : [Saladage,2013].

A10.2LES SGBD LES PLUS POPULAIRES

Le site http://db-engines.com/en/ranking[1] classe chaque mois les SGBD (et plus généralement les *gestionnaires de données*) les plus populaires, par classe de modèles et par popularité décroissante. Cette dernière est mesurée par le nombre de mentions dans les sites web, les forums, les réseaux sociaux, offres d'emploi, etc.[2] En mai 2015, les SGBD se répartissaient comme suit :

- **Content store** (2) : Jackrabbit, ModeShape
- **Document store** (28) : MongoDB, CouchDB, Couchbase, RavenDB, GemFire, Cloudant, RethinkDB, Datameer, Datomic, Mnesia, Microsoft Azure DocumentDB, PouchDB, Google Cloud Datastore, CloudKit, TokuMX, Clusterpoint, Terrastore, DensoDB, Djondb, EJDB, FleetDB, JasDB, LokiJS, RaptorDB, SenseiDB, Sequoiadb, SisoDb, WhiteDB
- **Event Store** (3) : InfluxDB, Event Store, NEventStore
- **Graph DBMS** (10) : Neo4j, Titan, Sparksee, Giraph, InfiniteGraph, InfoGrid, FlockDB, HyperGraphDB, GraphBase, VelocityGraph
- **Key-value store** (44) : Redis, Memcached, Riak, Ehcache, Hazelcast, Berkeley DB, Oracle Coherence, Amazon SimpleDB, Aerospike, Oracle NoSQL, Infinispan, LevelDB, GridGain, ZODB, GT.M, Tokyo Cabinet, NCache, WebSphere eXtreme Scale, WiredTiger, Tokyo Tyrant, XAP , Project Voldemort, Hibari, RocksDB, MapDB, STSdb, Scalaris, Kyoto Cabinet, Elliptics, HyperDex, Hamsterdb, ScaleOut StateServer, Bangdb, BergDB, CodernityDB, HyperLevelDB, Kyoto Tycoon, LedisDB, LightCloud, Nanolat, Quasardb, Resin Cache, Tarantool, TomP2P
- **Multi-model** (14) : Amazon DynamoDB, MarkLogic, OrientDB, Virtuoso, FoundationDB, ArangoDB, Sqrrl, Crate.IO, GraphDB, Amisa Server, Blazegraph, CortexDB, GlobalsDB, OrigoDB
- **Multivalue DBMS** (10) :, Adabas, UniData, UniVerse, D3, jBASE, Model 204, Northgate Reality, SciDB, OpenInsight, Rasdaman, OpenQM
- **Native XML DBMS** (4) : Sedna, BaseX, Tamino, eXist-db
- **Navigational DBMS** (2) : IMS, IDMS
- **Object oriented DBMS** (15) : Caché, Db4o, Versant Object Database, ObjectStore, Objectivity/DB, Perst, ObjectDB, GemStone/S, Eloquera, Siaqodb, Jade, Versant FastObjects, Starcounter, VelocityDB, WakandaDB
- **RDF store** (14) : Jena, Sesame, AllegroGraph, Algebraix, Stardog, Redland, 4store, RedStore, Strabon, BrightstarDB, CubicWeb, Dydra, Mulgara, SparkleDB

1. Les données de cette section sont publiée avec l'autorisation du responsable de ce site.
2. Cette notion de popularité est à interpréter avec précaution. On peut imaginer qu'un SGBD peu répandu mais particulièrement délicat à utiliser sera jugé plus populaire qu'un SGBD plus largement utilisé mais ne posant pas de problèmes.

- **Relational DBMS** (98) : Oracle, MySQL, Microsoft SQL Server, PostgreSQL, DB2, Microsoft Access, SQLite, SAP Adaptive Server, Teradata, FileMaker, Hive, Informix, SAP HANA, MariaDB, Firebird, Netezza, Microsoft Azure SQL Database, Vertica, dBASE, Ingres, Greenplum, SAP SQL Anywhere, Amazon Redshift, Interbase, SAP IQ, Impala, HyperSQL, mSQL, Derby, H2, Google BigQuery, MaxDB, SAP Advantage Database Server,TimesTen, OpenEdge, Teradata Aster, EnterpriseDB, Drizzle, VoltDB, Percona Server, ParAccel, Infobright, SQLBase, Oracle Rdb, MemSQL, MonetDB, DataEase, Empress, NuoDB, Red Brick, Amazon Aurora, Kdb+, Apache Drill, Altibase, NonStop SQL, R:BASE, Datacom/DB, solidDB, Clustrix, Vectorwise, 1010data, TokuDB, DBISAM, Pervasive PSQL, InfiniDB, Kognitio, FrontBase, OpenBase, VistaDB, Hadapt, NexusDB, EXASolution, Rainstor, Cubrid, ITTIA, eXtremeDB, ScimoreDB, Splice Machine, Akiban, LucidDB, XtremeData, Mimer SQL, WebScaleSQL, Dataupia, ScaleBase, ScaleDB, SQL.JS, GenieDB, Transbase, TransLattice, Tajo, SmallSQL, ElevateDB, c-treeACE, JethroData, JustOneDB, Postgres-XL, Valentina Server
- **Search engine** (15) : Solr, Elasticsearch, Splunk, Sphinx, Endeca, Google Search Appliance, Amazon CloudSearch, Microsoft Azure Search, Xapian, Indica, Compass, SearchBlox, Srch², DBSight, Exorbyte
- **Wide column store** (4) : Cassandra, HBase, Accumulo, Hypertable

Le recensement compte **263** gestionnaires. Les dix gestionnaires les plus populaires sont à ce moment les suivants :

Rang	Nom	Classe	Score
1	Oracle	Relationnel	1442.
2	MySQL	Relationnel	1294
3	SQL Server	Relationnel	1131
4	MongoDB	Document	277
5	PostgreSQL	Relationnel	274
6	DB2	Relationnel	201
7	Access	Relationnel	146
8	Cassandra	Colonnes	107
9	SQLite	Relationnel	105
10	Redis	Clé-valeur	95

Popularité par classe :

Classe	Score
Relationnel	82,6%
Document	6,2%
Moteurs de recherche	3,6%
Clé-valeur	3,2%
Colonnes	2,8%
Graphe	0,6%
XML natif	0,3%
RDF	0,3%
Multivaleurs	0,2%

Quelques observations

– La variété et le nombre de SGBD sont impressionnants, surtout compte tenu de la tendance naturelle du marché à la standardisation, qui, par nature, conduit à un appauvrissement de l'offre.

– La classe des systèmes relationnels est très majoritaire, tant en nombre de représentants (37%) qu'en popularité (82,6%).

– Parmi les SGBD NoSQL, c'est la classe des SGBD *orientés documents* qui occupe la deuxième place, et particulièrement MongoDB, en 4e position absolue.

– Parmi les SGBD NoSQL toujours, c'est la population des SGBD *Clé-valeurs* qui est la plus nombreuse, malgré la pauvreté du modèle de données.

– 164 SGBD (62%) n'atteignent pas le score de 1 (pour rappel, celui d'Oracle est de 1442); 71 n'atteignent pas le score de 0,1 et 44 n'atteignent pas le score de 0,01.

Les données de ranking au mois de mai 2015 sont disponibles sous la forme d'une base de données dans fichier **DBMS-Ranking.sql**.[3]

A10.3LA NOTATION XML

<A rédiger>

3. Mise à disposition avec l'autorisation de responsable du site.

A10.4LA NOTATION JSON

<A rédiger>

A10.5LE CONCEPT DE SCHÉMA DANS LES NOUVEAUX MODÈLES DE BASES DE DONNÉES

<A rédiger>

A10.6REPRÉSENTATION DE DOCUMENTS COMPLEXES DANS UNE BASE DE DONNÉES RELATIONNELLE

<A rédiger>

A10.7CONVERSION DE MODÈLES

L'existence, souvent dans la même organisation, de plusieurs modèles de bases de données pose un problème intéressant, celui de la conversion de données d'un modèle dans un autre. Ce problème n'est pas nouveau mais prend un accent tout particulier avec la disponibilité des SGBD NoSQL.

Dans cette section et dans les suivantes nous étudierons le processus de conversion tout en restant dans le contexte du modèle relationnel. Partant du format habituel de table, nous exprimerons un ensemble de modèles alternatifs en SQL et nous développerons les scripts de conversion. La traduction de ces données transformées selon l'un ou l'autre des SGBD devient alors une tâche triviale. Le modèle SQL joue dans cette approche le rôle de *modèle pivot*.

Cette manière de traiter le sujet est à but essentiellement didactique. Dans la pratique, la conversion se fera plus directement, l'extraction de données relationnelles conduisant au chargement des données dans le modèle cible sans étape intermédiaire.

Nous aborderons successivement les conversions entre le format standard de table et divers modéles généralement qualifés de *schema-less*.

A10.8LE MODÈLE DE TABLE UNIVERSELLE

Le terme de *table universelle* n'existe pas. Nous l'adopterons en hommage au concept théorique de *relation universelle* dont il s'inspire. La relation universelle **U** d'une base de données relationnelle **B** est une relation (fictive) telle que chaque rela-

tion de **B** peut être obtenue par une projection de **U**. **U** contient toutes les données et toutes des dépendences fonctionnelles de **B**, et elles seulement.[4]

Dans notre contexte, nous appellerons table universelle **U** d'une base de données **B** une table jouissant des mêmes propriétés, c'est-à-dire en particulier qu'à tout instant, le contenu de chaque table de **B** est égal à une projection de **U**.

Il existe plusieurs manière de définir une table universelle, par exemple par jointure (externe) de toutes les tables, par produit relationnel ou par l'union de ces tables. Nous choisirons cette dernière.

Signalons que tous les scripts relatifs au modèle de table universelle sont disponibles sous la forme du fichier **Table-Universelle.sql**.

A10.8.1Structures de la table universelle

La table universelle de la base de données CLICOM.db est structurée selon le schéma A10.1 et son contenu est illustré par la figure A10.3. Chaque ligne contient les données d'une ligne d'une table source. Chaque colonne représente une colonne d'une table source. On observe cependant que les colonnes de même nom (NCLI, NCOM, NPRO n'ont qu'un seul représentant dans la table universelle. On fait en effet l'hypothèse que si deux colonnes portent le même nom, elle représentent le même objet du domaine d'application. La table universelle est dotée d'un identifiant primaire technique EID sans signification. Toutes les colonnes sont facultatives à l'exception de l'identifiant EID.

Cette version du modèle ne distingue pas le type de l'entité représentée par une ligne (on qualifiera cette table de *non typée*). On peut donc à tout moment insérer une ligne constituée d'un ensemble arbitraire de valeurs. Si une colonne manque pour insérer une nouvelle ligne, elle est ajoutée par une instruction `alter table add column`. La table universelle offre une souplesse très proche de celle du *modèle historique* de Cassandra.

4. Ce concept a été très controversé (voir http://infolab.stanford.edu/jdu-symposium/talks/mendelzon.pdf). Présenté par certains auteurs comme une alternative au modèle relationnel, il a été contesté par d'autres, le trouvant trop éloigné des concepts modélisés (une table = un type d'entités). Il y a cependant quelques implémentations exploitant cette idée : Lotus Notes (IBM) et SESAM, un SGBD diffusé par Siemens puis repris par Fujtsu. La technique présentée dans cette section est très proche du modèle physique de SESAM.

```
createOrReplaceDB CLICOM-U;
create table CLICOM(
   EID       integer not null primary key autoincrement,
   NCLI      char(10),
   NOM       char(32),
   ADRESSE   char(60),
   LOCALITE  char(30),
   CAT       char(2),
   COMPTE    decimal(9,2),
   NPRO      char(15),
   LIBELLE   char(60),
   PRIX      integer,
   QSTOCK    integer,
   NCOM      char(12),
   DATECOM   date,
   QCOM      integer);
closeDB;
```

Figure A10.1 - Schema de la table universelle de CLICOM.db (version non typée)

On pourrait critiquer ce modèle en arguant une perte importante de place, toute ligne comportant une valeur pour chaque colonne, qu'elle soit pertinente ou non. Il n'en est rien. Le SGBD représente la présence ou l'absence d'une valeur d'une colonne facultative par un unique bit. La table CLICOM comporte 13 colonnes facultatives, ce qui se traduit par 13 bits de présence par ligne. Le surcoût est donc négligeable. D'autre part, la conversion des quatre tables sources n'entraîne pas de redondance autre que celle dont les données sources auraient été le siège.

Une variante de table universelle comporte une colonne additionnelle TYPE indiquent le type de l'entité représentée par chaque ligne. Ce modèle, qui reconnaît explicitement l'existence de types d'entités, est proche du modèle CQL de Cassandra. Il est évidemment plus contraignant puisque l'insertion d'une ligne doit respecter, d'une part, les colonnes autorisées pour le type d'entités choisi, et d'autre part les contraintes d'unicité et référentielles. Son schéma est donné par le script A10.2 et son contenu par la figure A10.4.

```
createOrReplaceDB CLICOM-UNI.db;
create table CLICOM(
   EID       integer not null primary key autoincrement,
   TYPE      char(18) not null,
   NCLI      char(10),
   NOM       char(32),
   ADRESSE   char(60),
   LOCALITE  char(30),
   CAT       char(2),
   COMPTE    decimal(9,2),
   NPRO      char(15),
   LIBELLE   char(60),
   PRIX      integer,
   QSTOCK    integer,
   NCOM      char(12),
   DATECOM   date,
   QCOM      integer);
closeDB;
```

Figure A10.2 - Schema de la table universelle de CLICOM.db (version typée)

A10.8.2Les index

Les index doivent favoriser les requêtes et la validation des contraintes propres aux données sources. La cohabitation de lignes d'origines différentes suggère l'utilisation d'index partiels pour soutenir l'accès par les identifiants sources :

```
create unique index CLI_NCLI
on CLICOM(NCLI) where TYPE = 'CLI';
```

EID	NCLI	NOM	ADRESSE	LOCALITE	CAT	COMPTE	NCOM	DATECOM	QCOM	NPRO	LIBELLE	PRIX	QCTOCK
1	B112	HANSENNE	23, r. Dumont	Poitiers	C1	1250	--	--	--	--	--	--	--
2	C123	MERCIER	25, r. Lemaître	Namur	C1	-2300	--	--	--	--	--	--	--
3	B332	MONTI	112, r. Neuve	Genève	B2	0	--	--	--	--	--	--	--
4	F010	TOUSSAINT	5, r. Godefroid	Poitiers	C1	0	--	--	--	--	--	--	--
5	K111	VANBIST	180, r. Florimont	Lille	B1	720	--	--	--	--	--	--	--
6	S127	VANDERKA	3, av. des Roses	Namur	C1	-4580	--	--	--	--	--	--	--
7	B512	GILLET	14, r. de l'Eté	Toulouse	B1	-8700	--	--	--	--	--	--	--
8	B062	GOFFIN	72, r. de la Gare	Namur	B2	-3200	--	--	--	--	--	--	--
9	C400	FERARD	65, r. du Tertre	Poitiers	B2	350	--	--	--	--	--	--	--
10	C003	AVRON	8, ch. de la Cure	Toulouse	B1	-1700	--	--	--	--	--	--	--
11	K729	NEUMAN	40, r. Bransart	Toulouse	--	0	--	--	--	--	--	--	--
12	F011	PONCELET	17, Clôs des Erables	Toulouse	B2	0	--	--	--	--	--	--	--
13	L422	FRANCK	60, r. de Wépion	Namur	C1	0	--	--	--	--	--	--	--
14	S712	GUILLAUME	14a, ch. des Roses	Paris	B1	0	--	--	--	--	--	--	--
15	D063	MERCIER	201, bvd du Nord	Toulouse	--	-2250	--	--	--	--	--	--	--
16	F400	JACOB	78, ch. du Moulin	Bruxelles	C2	0	--	--	--	--	--	--	--
17	K111	--	--	--	--	--	30178	2015-12-21	--	--	--	--	--
18	C400	--	--	--	--	--	30179	2015-12-22	--	--	--	--	--
19	S127	--	--	--	--	--	30182	2015-12-23	--	--	--	--	--
20	C400	--	--	--	--	--	30184	2015-12-23	--	--	--	--	--
21	F011	--	--	--	--	--	30185	2016-01-02	--	--	--	--	--
22	C400	--	--	--	--	--	30186	2016-01-02	--	--	--	--	--
23	B512	--	--	--	--	--	30188	2016-01-03	--	--	--	--	--
24	--	--	--	--	--	--	30178	--	25	CS464	--	--	--
25	--	--	--	--	--	--	30179	--	20	PA60	--	--	--
26	--	--	--	--	--	--	30179	--	60	CS262	--	--	--
27	--	--	--	--	--	--	30182	--	30	PA60	--	--	--
28	--	--	--	--	--	--	30184	--	120	CS464	--	--	--
29	--	--	--	--	--	--	30184	--	20	PA45	--	--	--
30	--	--	--	--	--	--	30185	--	15	PA60	--	--	--
31	--	--	--	--	--	--	30185	--	600	PS222	--	--	--
32	--	--	--	--	--	--	30185	--	260	CS464	--	--	--
33	--	--	--	--	--	--	30186	--	3	PA45	--	--	--
34	--	--	--	--	--	--	30188	--	70	PA60	--	--	--
35	--	--	--	--	--	--	30188	--	92	PH222	--	--	--
36	--	--	--	--	--	--	30188	--	180	CS464	--	--	--
37	--	--	--	--	--	--	30188	--	22	PA45	--	--	--
38	--	--	--	--	--	--	--	--	--	CS262	RAFT. PINE 200x6x2	75	45
39	--	--	--	--	--	--	--	--	--	CS264	RAFT. PINE 200x6x4	120	2690
40	--	--	--	--	--	--	--	--	--	CS464	RAFT. PINE 400x6x4	220	450
41	--	--	--	--	--	--	--	--	--	PA45	NAILS STEEL 45 (1K)	105	580
42	--	--	--	--	--	--	--	--	--	PA60	NAILS STEEL 60 (1K)	95	134
43	--	--	--	--	--	--	--	--	--	PH222	PL. BEECH 200x20x2	230	782
44	--	--	--	--	--	--	--	--	--	PS222	PL. PINE 200x20x2	185	1220

Figure A10.3 - Table universelle non typée de la base de données CLICOM.db

EID	TYPE	NCLI	NOM	ADRESSE	LOCALITE	CAT	COMPTE	NCOM	DATECOM	QCOM	NPRO	LIBELLE	PRIX	QCTOCK
1	CLI	B112	HANSENNE	23, r. Dumont	Poitiers	C1	1250	--	--	--	--	--	--	--
2	CLI	C123	MERCIER	25, r. Lemaître	Namur	C1	-2300	--	--	--	--	--	--	--
3	CLI	B332	MONTI	112, r. Neuve	Genève	B2	0	--	--	--	--	--	--	--
4	CLI	F010	TOUSSAINT	5, r. Godefroid	Poitiers	C1	0	--	--	--	--	--	--	--
5	CLI	K111	VANBIST	180, r. Florimont	Lille	B1	720	--	--	--	--	--	--	--
6	CLI	S127	VANDERKA	3, av. des Roses	Namur	C1	-4580	--	--	--	--	--	--	--
7	CLI	B512	GILLET	14, r. de l'Eté	Toulouse	B1	-8700	--	--	--	--	--	--	--
8	CLI	B062	GOFFIN	72, r. de la Gare	Namur	B2	-3200	--	--	--	--	--	--	--
9	CLI	C400	FERARD	65, r. du Tertre	Poitiers	B2	350	--	--	--	--	--	--	--
10	CLI	C003	AYRON	8, ch. de la Cure	Toulouse	B1	-1700	--	--	--	--	--	--	--
11	CLI	K729	NEUMAN	40, r. Bransart	Toulouse	--	0	--	--	--	--	--	--	--
12	CLI	F011	PONCELET	17, Clos des Erables	Toulouse	B2	0	--	--	--	--	--	--	--
13	CLI	L422	FRANCK	60, r. de Wépion	Namur	C1	0	--	--	--	--	--	--	--
14	CLI	S712	GUILLAUME	14a, ch. des Roses	Paris	B1	0	--	--	--	--	--	--	--
15	CLI	D063	MERCIER	201, bvd du Nord	Toulouse	--	-2250	--	--	--	--	--	--	--
16	CLI	F400	JACOB	78, ch. du Moulin	Bruxelles	C2	0	--	--	--	--	--	--	--
17	COM	K111	--	--	--	--	--	30178	2015-12-21	--	--	--	--	--
18	COM	C400	--	--	--	--	--	30179	2015-12-22	--	--	--	--	--
19	COM	S127	--	--	--	--	--	30182	2015-12-23	--	--	--	--	--
20	COM	C460	--	--	--	--	--	30184	2015-12-23	--	--	--	--	--
21	COM	F011	--	--	--	--	--	30185	2016-01-02	--	--	--	--	--
22	COM	C400	--	--	--	--	--	30186	2016-01-02	--	--	--	--	--
23	COM	B512	--	--	--	--	--	30188	2016-01-03	--	--	--	--	--
24	DET	--	--	--	--	--	--	30178	--	25	CS464	--	--	--
25	DET	--	--	--	--	--	--	30179	--	20	PA60	--	--	--
26	DET	--	--	--	--	--	--	30179	--	60	CS262	--	--	--
27	DET	--	--	--	--	--	--	30182	--	30	PA60	--	--	--
28	DET	--	--	--	--	--	--	30184	--	120	CS464	--	--	--
29	DET	--	--	--	--	--	--	30184	--	20	PA45	--	--	--
30	DET	--	--	--	--	--	--	30185	--	15	PA60	--	--	--
31	DET	--	--	--	--	--	--	30185	--	600	PS222	--	--	--
32	DET	--	--	--	--	--	--	30185	--	260	CS464	--	--	--
33	DET	--	--	--	--	--	--	30186	--	3	PA45	--	--	--
34	DET	--	--	--	--	--	--	30188	--	70	PA60	--	--	--
35	DET	--	--	--	--	--	--	30188	--	92	PH222	--	--	--
36	DET	--	--	--	--	--	--	30188	--	180	CS464	--	--	--
37	DET	--	--	--	--	--	--	30188	--	22	PA45	--	--	--
38	PRO	--	--	--	--	--	--	--	--	--	CS262	RAFT. PINE 200x6x2	75	45
39	PRO	--	--	--	--	--	--	--	--	--	CS264	RAFT. PINE 200x6x4	120	2690
40	PRO	--	--	--	--	--	--	--	--	--	CS464	RAFT. PINE 400x6x4	220	450
41	PRO	--	--	--	--	--	--	--	--	--	PA45	NAILS STEEL 45 (1K)	105	580
42	PRO	--	--	--	--	--	--	--	--	--	PA60	NAILS STEEL 60 (1K)	95	134
43	PRO	--	--	--	--	--	--	--	--	--	PH222	PL. BEECH 200x20x2	230	782
44	PRO	--	--	--	--	--	--	--	--	--	PS222	PL. PINE 200x20x2	185	1220

Figure A10.4 - Table universelle typée de la base de données CLICOM.db

```
create unique index PRO_NPRO
on CLICOM(NPRO) where TYPE = 'PRO';

create unique index COM_NCOM
on CLICOM(NCLI) where TYPE = 'COM';

create unique index DET_NCOM_NPRO
on CLICOM(NCOM,NPRO) where TYPE = 'DET';
```

On fera de même pour les index des clés étrangères :

```
create index COM_NCLI
on CLICOM(NCLI) where TYPE = 'COM';

create index DET_NPRO
on CLICOM(NPRO) where TYPE = 'DET';
```

On écarte un index sur NCOM where TYPE = 'DET', qui serait préfixe de DET_NCOM_NPRO, et donc inutile (voir chapitre 14, règle des index préfixes).

A10.8.3Migration des données standard vers une table universelle

Le processus de migration des données d'une base de données standard vers sa table universelle est très simple. Il correspond au script A10.5 dans l'hypothèse où la table CLICOM et les tables sources appartiendraient à la même base de données. Si ces tables appartiennent à des bases de données différentes, on passera par un script intermédiaire composé de requêtes `insert into CLICOM`, générées à partir de la base de données CLICOM.db.

```
insert into CLICOM(TYPE,NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE)
select 'CLI',NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE from CLIENT;
insert into CLICOM(TYPE,NPRO,LIBELLE,PRIX,QCOM)
select 'PROD',NPRO,LIBELLE,PRIX,QCOM from PRODUIT;
insert into CLICOM(TYPE,NCOM,NCLI,DATECOM)
select 'COM',NCOM,NCLI,DATECOM from COMMANDE;
insert into CLICOM(TYPE,NCOM,NPRO,QCOM)
select 'DET',NCOM,NPRO,QCOM from DETAIL;
```

Figure A10.5 - Migration des données standard vers la table universelle typée

La table universelle permet d'éviter les auto-jointures que traduisent les jointures basées sur les clés étrangères des données sources, comme le montre le script A10.6, qui exploite le partage de la colonne NCOM par les lignes originaires de COMMANDE et celles issues de DETAIL.

```
select  NCOM,NCLI,DATECOM,NPRO,QCOM
from    CLICOM
where   NCOM = '30184';
```

Figure A10.6 - La table universelle permet d'éviter certaines jointures courantes

```
+-------+------+------------+-------+------+
| NCOM  | NCLI | DATECOM    | NPRO  | QCOM |
+-------+------+------------+-------+------+
| 30184 | C400 | 2015-12-23 | --    | --   |
| 30184 | --   | --         | CS464 | 120  |
| 30184 | --   | --         | PA45  | 20   |
+-------+------+------------+-------+------+
```

Cette requête pourra s'appuyer sur l'index suivant :

```
create index COMDET_NCOM
on CLICOM(NCOM) where TYPE in ('COM','DET');
```

Pour les mêmes raisons, on créera les deux index suivants :

```
create index CLICOM_NCLI
on CLICOM(NCLI) where TYPE in ('CLI','COM');

create index PRODET_NCPRO
on CLICOM(NPRO) where TYPE in ('PRO','DET');
```

La technique n'est pas sans rappeler la structure des *clusters* d'Oracle (chapitre 4).

A10.8.4Expression des données sources

L'extraction des données sources à partir du contenu de la table universelle est triviale. On l'exprimera sous la forme de vues SQL.

```
create view V_CLIENT as
select  NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE
from    CLICOM  where TYPE = 'CLI';
```

Figure A10.7 - Expression des données sources à partir de la table universelle typée

A10.9LE MODÈLE ORIENTÉ COLONNES

Dans ce modèle, chaque propriété, qui dans le modèle standard est représentée par une colonne d'une table, est représentée par une table, dite *table-colonne*. Une table-colonne comporte généralement deux colonnes, l'une qui identifie une entité et l'autre qui spécifie la valeur de la propriété pour cette entité.

```
createOrReplaceDB CLICOM-COL.db;

create table CLI_NOM(
   NCLI      char(10) not null primary key,
   NOM       char(32) not null);
create table CLI_ADRESSE(
   NCLI      char(10) not null primary key,
   ADRESSE   char(60) not null);
create table CLI_LOCALITE(
   NCLI      char(10) not null primary key,
   LOCALITE  char(30) not null);
create table CLI_CAT(
   NCLI      char(10) not null primary key,
   CAT       char(2));
create table CLI_COMPTE(
   NCLI      char(10) not null primary key,
   COMPTE    decimal(9,2) not null);

create table PRO_LIBELLE(
   NPRO      char(15) not null primary key,
   LIBELLE   char(60) not null);
create table PRO_PRIX(
   NPRO      char(15) not null primary key,
   PRIX      integer  not null);
create table PRO_QSTOCK(
   NPRO      char(15) not null primary key,
   QSTOCK    integer  not null);

create table COM_NCLI(
   NCOM      char(12) not null primary key,
   NCLI      char(10) not null);
create table COM_DATECOM(
   NCOM      char(12) not null primary key,
   DATECOM   date     not null);

create table DET_QCOM(
   NCOM      char(12) not null,
   NPRO      char(15) not null,
   QCOM      integer  not null?
   primary key (NCOM,NPRO));

closeDB;
```

Figure A10.8 - Modèle orienté colonnes de la base de données CLICOM.db

Alors que le modèle classique représente un client par une ligne de la table CLIENT, le modèle orienté colonne le représente par les lignes extraites de chacune des cinq tables-colonnes enregistrant respectivement les noms, les adresses, les localités, les catégories et les comptes. Le script A10.8 crée les structures orientées colonnes de la base de données CLICOM.db.

Des index seront en tout cas créés pour chaque identifiant primaire et pour les colonnes de données susceptibles de faire l'objet de critères de sélection.

Nous avons utilisé les identifiants primaires comme identifiant d'entités. Cette transformation pose quelques petits problèmes :

– Quel est le sort des colonnes facultatives ? Deviennent-elles obligatoires dans le schéma final ou y restent-elles facultatives ? Dans le premier cas, aucune ligne n'apparaît dans la table-colonne pour les lignes sources n'ayant pas de valeur pour cette colonne. Une jointure avec une autre table-colonne doit être une jointure externe, afin de générer les valeurs *null* là où la seconde table ne possède pas de ligne. En outre, les identifiants d'entités de la table source seraient perdus si toutes les colonnes (hors identifiant primaire) étaient facultatives. Dans le second cas, une jointure naturelle suffira, toutes les tables-colonnes ayant le même nombre de lignes. C'est ce dernier cas qui est adopté dans la transformation.

– Les colonnes des identifiants primaires sont traitées de manière particulière puisqu'elle apparaissent dans toutes les tables-colonne sans disposer de leur propre table-colonne. Un identifiant composite conduit à des tables-colonnes non binaires (c'est le cas de DETAIL). Une structure alternative consisterait à utiliser un identifiant technique indépendant de manière à traiter chaque colonne de la même manière. Se poserait alors le problème de l'expression des identifiants composites et des clés étrangères composites. Formellement, ces contraintes devraient être exprimée sur des jointures qui réassemblent leurs composants. En pratique, elles se traduiront par des déclencheurs.

– Un schéma orienté colonnes comporte deux types de clés étrangères, celles qui assurent la cohésion des données de chaque entité (les valeurs de CLI_CAT.NCLI sont les valeurs de CLI_NOM.NCLI et inversement) et celles qui existaient parmi les tables sources (les valeurs de COM_NCLI.NCLI sont des valeurs de CLI_NOM.NCLI).

La figure A10.9 montre le contenu des tables CLI_NOM, CLI_LOCALITE et CLI_CAT.

NCLI	NOM
B112	HANSENNE
C123	MERCIER
B332	MONTI
F010	TOUSSAINT
K111	VANBIST
S127	VANDERKA
B512	GILLET
B062	GOFFIN
C400	FERARD
C003	AVRON

NCLI	LOCALITE
B112	Poitiers
C123	Poitiers
B332	Genève
F010	Poitiers
K111	Lille
S127	Namur
B512	Toulouse
B062	Namur
C400	Poitiers
C003	Toulouse

NCLI	CAT
B112	C1
C123	C1
B332	B2
F010	C1
K111	B1
S127	C1
B512	B1
B062	B2
C400	B2
C003	B1

```
| K729 | NEUMAN    |    | K729 | Toulouse  |    | K729 | --  |
| F011 | PONCELET  |    | F011 | Toulouse  |    | F011 | B2  |
| L422 | FRANCK    |    | L422 | Namur     |    | L422 | C1  |
| S712 | GUILLAUME |    | S712 | Paris     |    | S712 | B1  |
| D063 | MERCIER   |    | D063 | Toulouse  |    | D063 | --  |
| F400 | JACOB     |    | F400 | Bruxelles |    | F400 | C2  |
+------+-----------+    +------+-----------+    +------+-----+
```

Figure A10.9 - Trois des tables-colonnes issues de la table source CLIENT

Ce modèle présente l'avantage de réduire la taille des tables et les temps d'accès pour certaines requêtes simples (*quel est le nom du client C400* ?). Chaque table-colonne peut disposer des index qui lui sont le plus favorable. Le modèle permet aussi une évolution en souplesse du schéma, par ajout et suppression de tables-colonnes.

Remarque

Contrairement à ce que le nom *orienté-colonnes* pourrait laisser penser, ce modèle est différent de celui des SGBD NoSQL dits *orientés colonnes*, dans lesquels la notion de colonne est, comme nous l'avons vu, assez particulier.

A10.9.1Migration des données standard vers des tables-colonnes

Le script A10.10 charge les tables-colonnes à partir des tables standard.

```
insert into CLI_NOM      select NCLI,NOM       from CLIENT;
insert into CLI_ADRESSE  select NCLI,ADRESSE   from CLIENT;
insert into CLI_LOCALITE select NCLI,LOCALITE  from CLIENT;
insert into CLI_CAT      select NCLI,CAT       from CLIENT;
insert into CLI_COMPTE   select NCLI,COMPTE    from CLIENT;

insert into PRO_LIBELLE  select NPRO,LIBELLE   from PRODUIT;
insert into PRO_PRIX     select NPRO,PRIX      from PRODUIT;
insert into PRO_QSTOCK   select NPRO,QSTIOCK   from PRODUIT;

insert into COM_NCLI     select NCOM,NCLI      from COMMANDE;
insert into COM_DATECOM  select NCOM,NCLI      from COMMANDE;

insert into DET_QCOM     select NCOM,NPRO,QCOM from DETAIL;
```

Figure A10.10 - Migration des données standard vers les tables-colonnes

A10.9.2Expression des données sources

La reconstitution des données sources est réalisée par une jointure des tables-colonnes correspondantes (script A10.11).

```
create view V_CLIENT as
select  C1.NCLI,NOM,ADRESSE,LOCALITE,CAT,COMPTE
from    CLI_NOM C1,CLI_ADRESSE C2,CLI_LOCALITE C3,
        CLI_CAT C4,CLI_COMPTE C5
where   C1.NCLI = C2.NCLI and C1.NCLI = C3.NCLI
and     C1.NCLI = C4.NCLI and C1.NCLI = C5.NCLI;
```

Figure A10.11 - Expression des données sources à partir de la table universelle typée

Les scripts relatifs au modèle orienté colonnes sont disponibles sous la forme du fichier **Tables-colonnes.sql**.

A10.10LE MODÈLE CLÉ-VALEUR AGRÉGÉ

Les modèles de *table universelle* et de *tables-colonnes* partagent avec le modèle classique l'existence d'un schéma qui assigne aux tables et aux colonnes des noms qui évoquent de manière explicite les concepts du domaine d'application : CLIENT, ADRESSE, QSTOCK, etc.

Les modèles *Clé-valeur*, et leurs variantes *Attribut-valeur*, vont au contraire définir des structure de données de plus en plus génériques, c'est-à-dire plus indépendantes du domaine d'application.

Le modèle évoqué dans le titre (*Clé-valeur agrégé*) est de peu d'intérêt dans cette discussion. Nous le citons simplement parce qu'il correspond aux modèles NoSQL dits *clé-valeur.* Chaque entité est décrite par un identifiant primaire, souvent un simple nombre ou une adresse (URI par exemple) sans signification, accompagné d'une chaîne de caractères ou de bits sans structure (un CLOB ou un BLOB). L'extraction de fragments significatifs de cette chaîne est de la responsabilité des programmes d'application[5]. Nous n'en dirons donc rien de plus.

A10.11LE MODÈLE ATTRIBUT-VALEUR - VERSION 1

Rappelons d'abord que le terme *clé-valeur* désigne en toute généralité (c'est-à-dire au-delà des SGBD NoSQL) une forme d'expression de propriétés constituée du nom de la propriété et de la valeur qui lui est assignée. Ce format est très répandu dans une large variété d'applications. Citons par exemple les fichiers d'initialisation qui peuplent nos disques durs, et qui sont constitués d'une liste de couples clé-valeurs, tels que SQLfast.ini dont on donne un extrait :

```
autoclosedb = mainscript
select-align = Y
```

5. C'est le cas d'Oracle, qui applique à Berkeley DB un codage JSON.

```
csv-separator = ;
label-separator = |
outputtype = console
outputmode = write
```

Ce pattern est parfaitement applicable à la représentation du contenu d'une base de données. Pour un client particulier (C400 par exemple), on écrira NOM = FERARD. Plus généralement, on associera à l'entité C400 une liste de couples *clé-valeur* qui en précise les caractéristiques. Puisque nous allons nous limiter à ce domaine, nous adopterons un vocabulaire qui lui est plus naturel. Nous parlerons désormais de modèles *Attribut-valeur.*

Dans cette première version du modèle, les données traduites sous la forme Attribut-valeur sont enregistrées dans des tables qui correspondent chacune à une table du modèle standard. Nous les nommerons AV1_CLIENT, AV1_PRODUIT, AV1_COMMANDE et AV1_DETAIL.

Chaque table est constituée de trois colonnes dénommées Entite, Attribut et Valeur. Chaque ligne spécifie la valeur d'un l'attribut d'une entité. Par exemple, la table CV1_CLIENT contiendra la ligne (C400,NOM,FERARD) indiquant que le client C400 a un attribut NOM dont la valeur est FERARD.

La table AV1_CLIENT est définie par le script A10.12. La colonne Valeur est facultative pour permettre l'enregistrement des valeurs *null* de CAT. Les trois autres tables sont créées de manière similaire, hormis le caractère obligatoire de la colonne Valeur.

```
create table AV1_CLIENT(
   Entite    varchar(32) not null,
   Attribut  varchar(32) not null,
   Valeur    varchar(32),
   primary key (Entite,Attribut));
```

Figure A10.12 - Création de la table AV1_CLIENT

A10.11.1Migration des données standard vers des tables attribut-valeur

Le chargement des données dans les tables attribut-valeur à partir des tables standard est illustré par le script A10.13 montrant le cas des tables AV1_CLIENT et AV1_DETAIL.

La table AV1_DETAIL mérite un traitement un peu particulier. En effet, les entités *détail* sont identifiées par la colonne Entite, alors qu'elles sont, dans la table source DETAIL, identifiées par le couple (NCOM,NPRO). On convertit ce couple en une valeur unique de manière telle qu'il ne s'ensuit aucune ambiguïté, par exemple, si le caractère **-** ne peut apparaître dans les valeurs de NCOM et NPRO,

```
NCOM||'-'||NPRO
```

Remarquons que l'usage des valeurs des identifiants primaires pour désigner les entités (colonne Entité) est juste une simple facilité. Toute autre valeur, telle qu'un identifiant technique, conviendrait tout aussi bien.

```
insert into AV1_CLIENT (Entite,Attribut,Valeur)
       select NCLI,'NCLI',NCLI from CLIENT;
insert into AV1_CLIENT (Entite,Attribut,Valeur)
       select NCLI,'NOM',NOM from CLIENT;
insert into AV1_CLIENT (Entite,Attribut,Valeur)
       select NCLI,'ADRESSE',ADRESSE from CLIENT;
insert into AV1_CLIENT (Entite,Attribut,Valeur)
       select NCLI,'LOCALITE',LOCALITE from CLIENT;
insert into AV1_CLIENT (Entite,Attribut,Valeur)
       select NCLI,'CAT',CAT from CLIENT;
insert into AV1_CLIENT (Entite,Attribut,Valeur)
       select NCLI,'COMPTE',COMPTE from CLIENT;

. . .

insert into AV1_DETAIL (Entite,Attribut,Valeur)
       select NCOM||'-'||NPRO,'NCOM',NCOM from DETAIL;
insert into AV1_DETAIL (Entite,Attribut,Valeur)
       select NCOM||'-'||NPRO,'NPRO',NPRO from DETAIL;
insert into AV1_DETAIL (Entite,Attribut,Valeur)
       select NCOM||'-'||NPRO,'QCOM',QCOM from DETAIL;
```

Figure A10.13 - Migration des données vers les tables *attribut-valeur*

Ci-dessous des extraits des tables AV1_CLIENT et AV1_DETAIL.

Entite	Attribut	Valeur
B062	ADRESSE	72, r. de la Gare
B062	CAT	B2
B062	COMPTE	-3200
B062	LOCALITE	Namur
B062	NCLI	B062
B062	NOM	GOFFIN
B112	ADRESSE	23, r. Dumont
B112	CAT	C1
B112	COMPTE	1250
B112	LOCALITE	Poitiers
B112	NCLI	B112
B112	NOM	HANSENNE
B332	ADRESSE	112, r. Neuve
B332	CAT	B2
B332	COMPTE	0
B332	LOCALITE	Genève
B332	NCLI	B332
B332	NOM	MONTI
...	...	...

```
+-------------+----------+--------+
| Entite      | Attribut | Valeur |
+-------------+----------+--------+
| 30178-CS464 | NCOM     | 30178  |
| 30178-CS464 | NPRO     | CS464  |
| 30178-CS464 | QCOM     | 25     |
| 30179-CS262 | NCOM     | 30179  |
| 30179-CS262 | NPRO     | CS262  |
| 30179-CS262 | QCOM     | 60     |
| 30179-PA60  | NCOM     | 30179  |
| 30179-PA60  | NPRO     | PA60   |
| 30179-PA60  | QCOM     | 20     |
| ...         | ...      | ...    |
+-------------+----------+--------+
```

A10.11.2Expression des données sources

La reconstitution des données d'une table source est réalisée par une auto-jointure multiple de la table attribut-valeur correspondantes (script A10.14).

```
create view V_CLIENT as
select M1.Valeur as NCLI, M2.Valeur as NOM,
       M3.Valeur as ADRESSE,
       M4.Valeur as LOCALITE,
       M5.Valeur as CAT,
       cast(M6.Valeur as real) as COMPTE
from   AV1_CLIENT as M1, AV1_CLIENT as M2, AV1_CLIENT as M3,
       AV1_CLIENT as M4, AV1_CLIENT as M5, AV1_CLIENT as M6
where  M2.Entite = M1.Entite and M3.Entite = M1.Entite
and    M4.Entite = M1.Entite and M5.Entite = M1.Entite
and    M6.Entite = M1.Entite
and    M1.Attribut = 'NCLI'
and    M2.Attribut = 'NOM'
and    M3.Attribut = 'ADRESSE'
and    M4.Attribut = 'LOCALITE'
and    M5.Attribut = 'CAT'
and    M6.Attribut = 'COMPTE';
```

Figure A10.14 - Expression des données de la table source CLIENT à partir de la table attribut-valeur (première technique)

Le processus de construction d'une table multicolonnes (CLIENT) à partir de données désagrégées (AV1_CLIENT) n'est en réalité pas une nouveauté pour nous. La génération de tableaux étudiée à la section A8.1 de l'annexe A8 était un premier exemple de cette manipulation. Le script A10.14 est d'ailleurs très similaire au script A8.3. Nous avions à cette occasion suggéré une autre technique de recomposition des données sources basée sur un groupement des données de même valeur de la colonne Pro, qu'on pourrait ici assimiler à la colonne Entite. Le script A10.15 constitue cette seconde technique appliquée à la reconstruction de la table source CLIENT.

```
create view V_CLIENT as
select
max(case when Attribut = 'NCLI'     then Valeur end) as NCLI,
max(case when Attribut = 'NOM'      then Valeur end) as NOM,
max(case when Attribut = 'ADRESSE'  then Valeur end) as ADRESSE,
max(case when Attribut = 'LOCALITE' then Valeur end) as LOCALITE,
max(case when Attribut = 'CAT'      then Valeur end) as CAT,
max(case when Attribut = 'COMPTE'   then cast(Valeur as real)
         else -999999.9 end) as COMPTE
from  AV1_CLIENT
group by Entite;
```

Figure A10.15 - Expression des données de la table source CLIENT à partir de la table attribut-valeur (seconde technique)

Ce modèle offre un avantage évident : la souplesse d'évolution des attributs des entités. Un utilisateur peut à tout instant ajouter un nouvel attribut à une entité sans qu'il soit nécessaire de modifier le schéma de la table dédiée au type de cette entité. Les entités d'un même type n'ont pas nécessairement les mêmes attributs.

Les scripts relatifs à ce modèle attribut-valeur sont disponibles sous la forme du fichier **Attribut-valeur-v1.sql**.

A10.12LE MODÈLE ATTRIBUT-VALEUR - VERSION 2

Dans ce modèle, on élimine complètement la notion de schéma explicite, non seulement pour ce qui concerne les colonnes comme dans version 1, mais aussi la notion de table, qui regroupe les lignes des entités de même type. Ici, tous les triplets (Entité,Attribut,Valeur), quel soit le type de l'entité qu'ils représentent, sont rangés dans l'unique table **CLICOM** de la base de données.

Le type de l'entité concernée par un triplet sera indiqué par une colonne supplémentaire dénommée **TypeE**. Les triplets deviennent donc ... des quadruplets. Le script A10.16 définit la table CLICOM.

```
create table CLICOM(
   TypeE     varchar(32) not null,
   Entite    varchar(32) not null,
   Attribut  varchar(32) not null,
   Valeur    varchar(256),
   primary key (Type,Entite,Attribut));
```

Figure A10.16 - Création de la table CLICOM

Ci-dessous des extraits des tables AV1_CLIENT et AV1_DETAIL.

TypeE	Entite	Attribut	Valeur
CLIENT	B062	ADRESSE	72, r. de la Gare
CLIENT	B062	CAT	B2
CLIENT	B062	COMPTE	-3200
CLIENT	B062	LOCALITE	Namur
CLIENT	B062	NCLI	B062
CLIENT	B062	NOM	GOFFIN
CLIENT	B112	ADRESSE	23, r. Dumont
CLIENT	B112	CAT	C1
CLIENT	B112	COMPTE	1250
CLIENT	B112	LOCALITE	Poitiers
CLIENT	B112	NCLI	B112
CLIENT	B112	NOM	HANSENNE
CLIENT	B332	ADRESSE	112, r. Neuve
CLIENT	B332	CAT	B2
CLIENT	B332	COMPTE	0
CLIENT	B332	LOCALITE	Genève
CLIENT	B332	NCLI	B332
CLIENT	B332	NOM	MONTI
...	...	...	...
DETAIL	30178-CS464	NCOM	30178
DETAIL	30178-CS464	NPRO	CS464
DETAIL	30178-CS464	QCOM	25
DETAIL	30179-CS262	NCOM	30179
DETAIL	30179-CS262	NPRO	CS262
DETAIL	30179-CS262	QCOM	60
DETAIL	30179-PA60	NCOM	30179
DETAIL	30179-PA60	NPRO	PA60
DETAIL	30179-PA60	QCOM	20
...	...	...	...

La migration des données des tables sources vers la table CLICOM s'effectue via un script très semblable à celui de la version 1, auquel on ajoute les valeurs de la colonne TypeE. Il en est de même de la reconstitution des données sources à partir de la table CLICOM.

Dans cette version, la prise en compte d'entités d'un nouveau type ne nécessite plus la création d'une nouvelle table mais simplement d'une nouvelle valeur de TypeE.

Les scripts relatifs à la version 2 du modèle attribut-valeur sont disponibles sous la forme du fichier **Attribut-valeur-v2.sql**.

A10.13LE MODÈLE ATTRIBUT-VALEUR - VERSION 3

Cette troisième version du modèle attribut-valeur reprend l'idée de la deuxième version mais exprime de manière différente l'information sur le type des entités. Ce type n'y est plus représenté d'une manière spéciale par la colonne TypeE, mais comme un simple attribut de l'entité. On assigne donc à chaque entité un nouvel attribut de nom **TypeE** dont la valeur (**Valeur**) indique le type. Comme les identifiants primaires n'identifient pas les entités indépendamment de leur type (on peut

imaginer un client et un produit possédant la même valeur de leur identifiant), nous devrions compléter les valeurs de la colonne Entite en les préfixant d'un code unique pour ce type (par exemple **`CLIENT_C400`** ou **`CLI_C400`** pour l'entité client C400.

Nous allons cependant procéder autrement, en utilisant un *identifiant technique* de type `integer` dont nous génèrerons nous-mêmes les valeurs. C'est ce que traduit le script A10.17. Le script A10.18 charge dans cette table les données de la table source CLIENT. De manière à attribuer à l'identifiant primaire des entiers consécutifs, chaque série de chargement initialise la variable **`entite`** à la dernière valeur enregistrée.

```
create table CLICOM(
   Entite    integer     not null,
   Attribut  varchar(32) not null,
   Valeur    varchar(256),
   primary key (Entite,Attribut));
```

Figure A10.17 - Création de la table CLICOM

```
extract entite = select coalesce(max(Entite),0) from CLICOM;

for cli,nom,adr,loc,cat,cpt = [select * from CLIENT];
  compute entite = $entite$ + 1;
  function nom = LStr:SQLquote2 {$nom$};
  function adr = LStr:SQLquote2 {$adr$};
  function loc = LStr:SQLquote2 {$loc$};
  insert into CLICOM values($entite$,'TypeE','CLIENT');
  insert into CLICOM values($entite$,'NCLI','$cli$');
  insert into CLICOM values($entite$,'NOM','$nom$');
  insert into CLICOM values($entite$,'ADRESSE','$adr$');
  insert into CLICOM values($entite$,'LOCALITE','$loc$');
  insert into CLICOM values($entite$,'CAT',
                            case when '$cat$' = ''
                            then null else '$cat$' end);
  insert into CLICOM values($entite$,'COMPTE',$cpt$);
endfor;
```

Figure A10.18 - Migration des données sources de la table CLIENT vers la table CLICOM

Ci-dessous un extrait de la table CLICOM comprenant toutes les données des tables sources :

```
+--------+----------+----------------------+
| Entite | Attribut | Valeur               |
+--------+----------+----------------------+
| 1      | TypeE    | CLIENT               |
| 1      | ADRESSE  | 23, r. Dumont        |
| 1      | CAT      | C1                   |
| 1      | COMPTE   | 1250                 |
| 1      | LOCALITE | Poitiers             |
| 1      | NCLI     | B112                 |
```

```
| 1      | NOM      | HANSENNE             |
| 2      | TypeE    | CLIENT               |
| 2      | ADRESSE  | 25, r. Lemaître      |
| 2      | CAT      | C1                   |
| 2      | COMPTE   | -2300                |
| 2      | LOCALITE | Namur                |
| 2      | NCLI     | C123                 |
| 2      | NOM      | MERCIER              |
| 3      | TypeE    | CLIENT               |
| 3      | ADRESSE  | 112, r. Neuve        |
| 3      | CAT      | B2                   |
| 3      | COMPTE   | 0                    |
| 3      | LOCALITE | Genève               |
| 3      | NCLI     | B332                 |
| 3      | NOM      | MONTI                |
| ...    | ...      | ...                  |
| 31     | TypeE    | DETAIL               |
| 31     | NCOM     | 30178                |
| 31     | NPRO     | CS464                |
| 31     | QCOM     | 25                   |
| 32     | TypeE    | DETAIL               |
| 32     | NCOM     | 30179                |
| 32     | NPRO     | PA60                 |
| 32     | QCOM     | 20                   |
| 33     | TypeE    | DETAIL               |
| 33     | NCOM     | 30179                |
| 33     | NPRO     | CS262                |
| 33     | QCOM     | 60                   |
| ...    | ...      | ...                  |
+--------+----------+----------------------+
```

La reconstruction de la table source CLIENT est réalisée par le script A10.19.

```
create view V_CLIENT as
select
max(case when Attribut = 'NCLI'     then Valeur end) as NCLI,
max(case when Attribut = 'NOM'      then Valeur end) as NOM,
max(case when Attribut = 'ADRESSE'  then Valeur end) as ADRESSE,
max(case when Attribut = 'LOCALITE' then Valeur end) as LOCALITE,
max(case when Attribut = 'CAT'      then Valeur end) as CAT,
max(case when Attribut = 'COMPTE' then cast(Valeur as real)
                                  else -999999.9 end) as COMPTE
from  CLICOM
where Entite in (select Entite from CLICOM
                 where  Attribut = 'TypeE'
                 and    Valeur = 'CLIENT')
group by Entite;
```

Figure A10.19 - Reconstruction de la table source CLIENT à partir de la table CLICOM

A10.14LE MODÈLE DE GRAPHE

Les modèles de graphes représentent les données sous la forme d'un graphe, c'est-à-dire d'un ensemble de noeuds et d'un ensemble d'arcs entre ces noeuds, un arc représentant une association entre deux noeuds. On assigne généralement un *type* à chaque noeud et à chaque arc, mais aussi, selon les besoins, d'autres attributs.

La base de données CLICOM.db se prête assez bien à une représentation sous la forme d'un graphe. Chaque entité constitue un noeud, chaque référence via une clé étrangère constitue un arc et chaque colonne un attribut des noeuds. Les clés étrangères disparaissent puisqu'elles sont désormais traduites par des arcs. Le problème des clés étrangères composites ne se pose donc plus.

De manière à conserver la souplesse d'évolution des modèles attribut-valeurs, nous organiserons les données selon trois tables :

– la table **ENTITE**, qui reprend l'ensemble des entités avec leur type (les *noeuds*)

– la table **ATTRIBUT**, qui associe à chaque entité ses attributs,

– la table **ASSOCIATION**, qui définit les couples d'entités et leur type.

La notion d'*attribut d'association* n'est pas reprise ici car elle n'est pas pertinente dans le modèle source. Le script A10.20 crée les tables du modèle de graphes.

```
create table ENTITE(
   Entite    integer     not null primary key,
   TypeE     varchar(32) not null);

create table ASSOCIATION(
   Source    integer not null,
   Cible     integer not null,
   TypeA     varchar(32) not null,
   primary key (TypeA,Source,Cible));

create table ATTRIBUT(
   Entite    integer     not null,
   Attribut  varchar(32) not null,
   Valeur    varchar(256),
   primary key (Entite,Attribut));
```

Figure A10.20 - Création des tables du modèle de graphes

Le chargement de la table ASSOCIATION nécessitera une correspondance entre l'identifiant primaire des tables sources et l'identifiant technique d'entité créé lors du chargement des données de la table ENTITE. Cette correspondance sera stockée dans la table temporaire IDENTITE, à supprimer après usage (script A10.21).

```
create temporary table IDENTITE(
   Entite    integer     not null,
   TypeE     varchar(32) not null,
   EID       varchar(32),
   primary key (TypeE,EID),
   unique(Entite) );
```

Figure A10.21 - Table de correspondance entre les identifiants relationnels et les identifiants d'entités

La conversion des tables en graphes est assez semblable à celle du modèle attribut-valeur, dont elle s'inspire (script A10.22). Elle présente trois diffférences :

- insertion d'une ligne dans la table ENTITE, représentant un nouveau noeud du graphe,
- insertion dans la table de correspondance IDENTITE du couple des identifiants,
- absence de traduction des clés étrangères

```
for com,cli,dat = [select * from COMMANDE];
  compute entite = $entite$ + 1;

  insert into ENTITE values($entite$,'COMMANDE');
  insert into IDENTITE values($entite$,'COMMANDE','$com$');

  insert into ATTRIBUT values($entite$,'NCOM','$com$');
  insert into ATTRIBUT values($entite$,'DATECOM','$dat$');
endfor;
```

Figure A10.22 - Chargement des entités de la table COMMANDE

La création des associations ne peut s'effectuer que lorsque tous les noeuds ont été créés. Le script A10.23 crée les associations entre chaque entité DETAIL et les entités COMMANDE et PRODUIT dont elle dépend. Les jointures permettent de retrouver, pour chaque entité DETAIL, son identifiant, celui de l'entité COMMANDE associée et celui de l'entité PRODUIT associée.

```
for det,com,pro =
    [select DI.Entite,CI.Entite,PI.Entite
     from   DETAIL D,IDENTITE DI,IDENTITE CI,IDENTITE PI
     where  D.NCOM||'-'||D.NPRO = DI.EID
     and    DI.TypeE = 'DETAIL'
     and    D.NCOM = CI.EID and CI.TypeE = 'COMMANDE'
     and    D.NPRO = PI.EID and PI.TypeE = 'PRODUIT'];

  insert into ASSOCIATION values($det$,$com$,'DETAIL(NCOM)');
  insert into ASSOCIATION values($det$,$pro$,'DETAIL(NPRO)');
endfor;
```

Figure A10.23 - Création des deux associations entre une entité DETAIL et ses entités COMMANDE et PRODUIT

Remarquons qu'on peut envisager une autre conversion sous la forme de graphe dans lequel chaque entité, mais aussi chaque valeur de colonne, constitue un noeud. Il existe alors deux sortes d'arcs : les arcs entre deux entités et les arcs entre une entité et une valeur d'attribut, ces derniers arcs portant le nom de l'attribut dont ils dérivent. Cette modélisation nous amène alors à celle des ontoilogies, telles que RDF et OWL

A10.15LES MODÈLES D'ONTOLOGIES

<à rédiger>

A10.16LE MODÈLE ORIENTÉ DOCUMENT SIMPLE

<à rédiger>

A10.17LE MODÈLE ORIENTÉ DOCUMENT COMPLEXE

<à rédiger>

A10.18INDEXATION DE DOCUMENTS COMPLEXES

<à rédiger>