L 'OBJET-RELATIONNEL

- **Présentation de la norme SQL3**
- **Support de SQL3 dans Oracle 8**
- **Les systèmes Objet-relationnel**

1. Evolution des SGBD relationnels

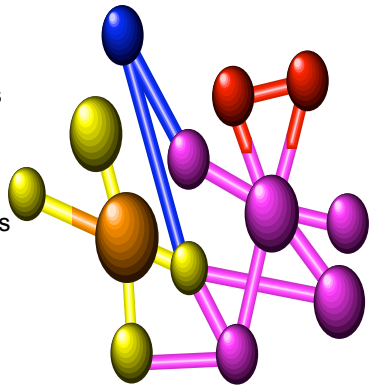
- **Années 80 :**
  - centrés sur le transactionnel
  - applications dites OLTP (One Line Transaction Processing)
- **Années 90 :**
  - développement du décisionnel
  - applications dites OLAP (One Line Analysis Processing)
  - intégration de données multi-dimensionnelle (cube 3D)
- **Années 95 :**
  - besoins croissants des applications avancées (CAO, AGL, SIG,…)
  - avènement des technologies objet et du Web

1. LA NORMALISATION

- **SQL2 [ISO92] :**
  - diversification des domaines :
    - dates, monnaies, le temps et les intervalles,
    - les chaînes de bits (LOBs, BLOBs, CLOBs)
  - meilleur support de l 'intégrité (clause CHECK)
  - intégration étendue de l 'algèbre relationnelle (union et jointure externes, meilleure gestion des valeurs nulles)
  - possibilité de SELECT en argument d 'un FROM
- **Des problèmes demeurent:**
  - absence de pointeurs visibles par l 'utilisateur
    - restaurer la navigation et le partage référentiel d 'objets
  - le non-support de domaines composés
    - non-structurés et ne supportent pas les recherches associatives
  - la non-intégration des opérations

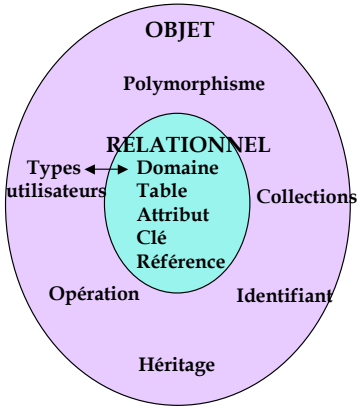
1. LE SUPPORT D 'OBJETS COMPLEXES

- Nécessité d'introduire des attributs multi-valués
- Offrir des collections prédéfinies telles que liste, ensemble, tableau, …
- Imbrication des collections pour représenter des objets très compliqués
- **Exemple** :
  - Molécule { list <Atome, Connexions>}
  - Atome { Noyau, list <Electrons> }



1. L 'OBJET-RELATIONNEL

- **Extension du modèle relationnel**
  - attributs multivalués : structure, liste, tableau, ensemble, ...
  - héritage sur relations et types
  - domaine type abstrait de données (structure cachée + méthodes)
  - identité d'objets
- **Extension de SQL**
  - définition des types complexes avec héritage
  - appels de méthodes en résultat et qualification
  - imbrication des appels de méthodes
  - surcharge d'opérateurs



1. SQL3

- **Objectifs**
  - supporter de manière intelligente des données multi-media
- **Solutions**
  - support d 'objets complexes (ADTs)
  - NF2 (Non Firts Normal Form) :
    - forme normale supportant des domaines multivalués
  - Modèle relationnel imbriqué (Nested Model)
    - un domaine peut lui-même être valué par des tables
- **Normalisation longue et difficile**
  - Committee Draft – 1/96
  - Draft International Standard – 12/98
  - International Standard – 7/99
- **Spécification volumineuse**
  - plus de 1500 pages

1. LES COMPOSANTS

- **Part 1: Framework**
  - description non-technique de comment le document est structuré.
- **Part 2: Foundation**
  - noyau de spécification, incluant les types de données abstraits.
- **Part 3: SQL/CLI**
  - interface d'appel client.
- **Part 4: SQL/PSM**
  - le langage de spécifications de procédures stockées
- **Part 5: SQL/Bindings**
  - les liens SQL dynamique et "embedded" SQL repris de SQL-92.
- **Part 6: SQL/XA**
  - spécification de l'interface XA pour moniteur transactionnel.
- **Part 7: SQL/Temporal**
  - support du temps dans SQL3

1. LES COMPOSANTS ...

- **Part 8: abandonné**
- **Part 9: SQL/MED**
  - utilisation de SQL pour accéder à des données non SQL.
- **Part 10: SQL/OBJ**
  - l 'utilisation de SQL depuis un langage objet et C++, Smalltalk ou Java

- **Au-delà de SQL3, vers SQL4 :**
  - **SQL/MM**
    - pour la spécification de types utilisateurs multi-media
  - **SQL/RDA**
    - pour la spécification du protocole de transfert des données entre le client et le serveur

1. LES PROCEDURES (PSM)

- **Langage de programmation de procédures**
  - déclaration de variables
  - assignation
  - conditionnels CASE, IF
  - boucles LOOP, FOR
  - exceptions SIGNAL, RESIGNAL
  - possibilité de procédures et fonctions externes
- **Possibilité de structuration en modules**

1. SQL3 - LES OBJETS

- **Extensibilité des types de données**
  - Définition de types abstraits
  - Possibilité de types avec ou sans OID
- **Support d'objets complexes**
  - Constructeurs de types (tuples, set, list, …)
  - Utilisation de référence (OID)
- **Héritage**
  - Définition de sous-types
  - Définition de sous-tables

1. LES TYPES ABSTRAITS

- **CREATE TYPE <nom ADT> <corps de l'ADT>**
- **<corps de l'ADT>**
  - <OID option> ::= WITH OID VISIBLE
    - objets sans OID par défaut
  - <subtype clause> ::= UNDER <supertype clause>
    - possibilité d'héritage multiple avec résolution explicite
  - <member list>
    - <column definition> : attributs publics ou privés
    - <function declaration> : opérations publiques
    - <operator name list> : opérateurs surchargés
    - <equals clause>, <less-than clause> : définition des ordres
    - <cast clause> : fonction de conversion de types

1. QUELQUES EXEMPLES

- **Un type avec référence**

```
CREATE TYPE WITH OID PHONE (
    country VARCHAR,
    area VARCHAR,
    number int,
    description CHAR(20))
```

- **Un type sans référence**

```
CREATE TYPE PERSON
    (nss INT,
    nom VARCHAR,
    tel phone)
```

- **Un sous-type**

```
CREATE TYPE STUDENT UNDER PERSON (
    major VARCHAR,
    year INT)
```

1. LES CONSTRUCTEURS DE TYPE

- **Les constructeurs de base (types paramétrés):**
  - collections : SET(T), MULTISET(T), LIST(T)

```
CREATE TYPE PERSON (
    nss INT,
    nom VARCHAR,
    prénoms LIST(varchar),
    tel SET(phone))
```

- **Les références**
  - possibilité de référencer un objet créé “without OID”

```
CREATE TYPE CAR (
    number CHAR(9),
    color VARCHAR,
    owner REF(person))
```

- **Les constructeurs additionnels**
  - stack, queue, array, insertable array (exemple : texte)
  - non intégrés dans le langage mais peuvent être ajoutés

1. LES FONCTIONS

- **Les fonctions peuvent être associées à une base, un type, une table,...**
- **Syntaxe :**

```
[<function type>] : CONSTRUCTOR, ACTOR, DESTRUCTOR
FUNCTION <function name> <parameter list>
RETURNS <function results> <SQL procedure> | <file name>
END FUNCTION
```

- **Exemple :**

```
CREATE FUNCTION sell (c Ref(Constructor), amount MONEY)
    UPDATE Constructor
    SET total = total + amount
    WHERE Ref(Constructor) = c
END FUNCTION
```

- **Langage de programmation**
  - SQL et SQL3 PSM, Langage externe

1. LES TABLES

- **Caractéristiques**
  - une table peut posséder des attributs d'un type abstrait
  - un tuple contient des références ou des valeurs complexes
  - un attribut peut être de type référence (REF <type> ou with OID)
- **Exemples**

```
CREATE TABLE Cars OF car ;        // utilisation d ’un type prédéfini

CREATE TABLE Constructors OF NEW TYPE Constructor (
    name VARCHAR,
    total MONEY) ;                // définition d ’un nouveau type

CREATE TABLE FrenchConstructors UNDER Constructors(
    taxe MONEY)                   // définition d ’une sous-table
```


1. L ’APPEL DE FONCTIONS ET D ’OPERATEURS

- **Appel de fonctions :**

```
SELECT r.name
FROM emp j, emp r
WHERE j.name = 'Joe'
AND distance(j.location,r.location) < 1 ;
```

- **Appel d'opérateurs ou prédicats :**

```
SELECT r.name
FROM emp e, emp r
WHERE e.name = 'Joe ’
AND contained(r.location, circle(e.location,1)) ;
```

1. LE PARCOURS DES REFERENCES

- **Les fonctions Ref et DeRef sont implicites :**

  CREATE TABLE cars OF TYPE car. ;
  SELECT c.Owner.name FROM cars c WHERE color = 'red ';

- **Possibilité de cascader la notation pointée :**

  SELECT dname FROM dept WHERE 1985 IN auto.years ;

- **Généralisation possible aux chemins multiples :**

  SELECT dname FROM dept
  WHERE autos.(year=1985 and name = 'Ford');

- **Toute collection peut être utilisée en place d'une table**

1. EXEMPLE DE TABLES IMBRIQUEES

Services

N°	Chef	Adresse	Employés		Dépenses		
			Nom	Age	NDep	Montant	Motif
24	Paul	Versailles	Pierre	45	1	2600	Livres
			Marie	37	2	8700	Mission
					3	15400	Portable
25	Patrick	Paris	Nom	Age	NDep	Montant	Motif
			Eric	42	5	3000	Livres
			Julie	51	7	4000	Mission

1. COMPARAISON AVEC LE RELATIONNEL

- **Accès en relationnel**

```
select effdate, name, vehicleyr
from policy, customers, vehicles
where policy.custno = customers.custno
and policy.vehicleno = vehicles.vehicleno
and model = 'ferrari';
```

- **Accès en objet-relationnel**

```
select p.effdate, p.name, p.vehicleyr
from policy p
where p.carmodel.make = 'ferrari';
```


L 'objet-relationnel dans Oracle 8

ADTs
References
Nested Tables
Large Objects (LOBs)

http://www-db.stanford.edu/~ullman/fcdb/oracle/or-objects.html

2. LES TYPES

Création d 'un type :

CREATE TYPE t AS OBJECT (
list of attributes and methods
);
/

/ pour qu'Oracle intègre la définition

Suppression d 'un type :

DROP TYPE t;

** toutes les tables contenant t doivent être détruites au préalable, sinon « t : still in used »*

```
CREATE TYPE PointType AS OBJECT(
        x NUMBER,
        y NUMBER );
    /

 CREATE TYPE LineType AS OBJECT(
        end1 PointType,
        end2 PointType);
    /

CREATE TABLE Lines (
        lineID INT,
        line   LineType);

INSERT INTO Lines
VALUES ( 27, LineType(
                        PointType(0.0, 0.0),
                        PointType(3.0, 4.0)));
```


2. LES METHODES

```
CREATE TYPE LineType AS OBJECT (
        end1 PointType,
        end2 PointType,
        MEMBER FUNCTION length(scale IN NUMBER) RETURN NUMBER,
        PRAGMA RESTRICT_REFERENCES(length, WNDS)  );
    /
```

- Il faut spécifier le mode de chaque argument : IN, OUT, INOUT
- **Pragma** : permet d 'indiquer à la BD
  - WNDS = write no database state
  - indispensable pour utiliser la méthode dans les requêtes

2. LES METHODES ...

```
CREATE TYPE BODY LineType AS
    MEMBER FUNCTION length(scale NUMBER) RETURN NUMBER IS
        BEGIN
            RETURN scale *
                SQRT((SELF.end1.x-SELF.end2.x)*(SELF.end1.x-SELF.end2.x) +
                    (SELF.end1.y-SELF.end2.y)*(SELF.end1.y-SELF.end2.y)
                );
        END;
END;
/
```

- **Self** : référence le tuple courant sur lequel est appliquée la méthode
- Le langage de définition est PL/SQL, (ou Java dans Oracle8i)

2. LES REQUETES

- **Utilisation d 'une méthode:**

```
SELECT lineID, ll.line.length(2.0)
FROM Lines ll;
```

- **Traversée de chemin:**

```
SELECT ll.line.end1.x, ll.line.end1.y
FROM Lines ll;
```

- **Valeur complexe en retour:**

```
SELECT ll.line.end2
FROM Lines ll;
```

** le résultat de cette requête serait PointType(3,4), le constructeur de type est utilisé pour afficher un résultat*

2. LES RELATIONS

- **Les types Oracle8 peuvent être utilisés :**
  - soit comme types abstraits de données (ADTs)
  - soit comme « RowType » selon SQL3

```
CREATE TABLE Lines1 OF LineType;
```

- équivalent à :

```
CREATE TABLE Lines1 (
  end1 PointType,
  end2 PointType );
```

- mais en plus on peut utiliser la méthode length :

```
SELECT AVG(ll.length(1.0))
FROM Lines1 ll;
```


2. L 'UTILISATION DES REFERENCES

```
CREATE TABLE Lines2 (
      end1 REF PointType,
      end2 REF PointType );
```

- REF : permet de créer une référence à partir d 'un objet tuple

```
CREATE TABLE Points OF PointType;

INSERT INTO Lines2
      SELECT REF(pp), REF(qq)
      FROM Points pp, Points qq
      WHERE pp.x < qq.x;
```

* restriction : pour manipuler des références, il faut que l 'objet soit un tuple
* INSERT INTO LInes2 VALUES(REF(PointType(1,2), REF(PointType(3,4)) n 'est pas possible car PointType(1,2) n 'appartient à aucune relation

```
SELECT ll.end1.x, DEREF(ll.end2)
FROM Lines2 ll;
```


2. LES TABLES IMBRIQUEES

- **Pour obtenir une relation comme type d 'un attribut, il faut définir un nouveau type :**

```
CREATE TYPE PolygonType AS TABLE OF PointType;
/

CREATE TABLE Polygons (
      name   VARCHAR2(20),
      points PolygonType)
      NESTED TABLE points STORE AS PointsTable;
```

2. LES TABLES IMBRIQUEES ...

```
INSERT INTO Polygons VALUES (
   ' square ', PolygoneType(
                  PointType(0.0,0.0),PointType(0.0,1.0),
                  PointType(1.0,0.0),PointType(1.0,1.0)));

SELECT  points
FROM  Polygons
WHERE  name = ' square ';

SELECT  ss.x
FROM  THE (SELECT points FROM Polygons WHERE name = ' square ') ss
WHERE  ss.x = ss.y;
```

2. RELATIONS IMBRIQUEES & REFERENCES

```
CREATE TYPE PolygonRefType AS TABLE OF REF PointType; /

CREATE TABLE PolygonsRef (
        name   VARCHAR2(20),
        pointsRef PolygonRefType)
        NESTED TABLE pointsRef STORE AS PointsRefTable;

SELECT ss.COLUMN_VALUE.x
FROM THE  (      SELECT pointsRef
            FROM PolygonsRef
            WHERE name = 'square' ) ss
WHERE ss.COLUMN_VALUE.x = ss.COLUMN_VALUE.y;
```

* Puisque la relation imbriquée n 'a pas de nom d 'attribut, alors Oracle fournit un nom par défaut **COLUMN_VALUE**

2. CONVERSION : RELATION -> OBJET-RELATION

- **Supposons une vieille relation de lignes plates :**

```
CREATE TABLE LinesFlat(
        id INT, x1 NUMBER, y1 NUMBER, x2 NUMBER, y2 NUMBER);
```

- **Transfert dans la relation Lines :**

```
INSERT INTO Lines
   SELECT id, LineType(PointType(x1,y1), PointType(x2,y2))
   FROM LinesFlat;
```


2. CONVERSION : RELATION -> OBJET-RELATION ...

- **Soit la relation représentant des polygones à plat :**

```
CREATE TABLE PolyFlat(
    name VARCHAR(20), x NUMBER, y NUMBER);
```

- **Transfert d 'un polygone dans la relation Polygons :**

```
INSERT INTO Polygons
  VALUES ('square',
            CAST(
                    MULTISET(       SELECT x, y
                                    FROM PolyFlat
                                    WHERE name = 'square')
                    AS PolygonType
             )
            );
```


2. CONVERSION : RELATION -> OBJET-RELATION ...

- **Transfert de tous les polygones dans la relation Polygons :**

```
INSERT INTO Polygons
  SELECT pp.name,
            CAST(
                    MULTISET(       SELECT x, y
                                    FROM PolyFlat    qq
                                    WHERE qq.name = pp.name )
                    AS PolygonType
            )
  FROM PolyFlat pp;
```

**Pb : insertion d 'autant de n-uplets Polygon que de n-uplets dans PolyFlat*

2. CONVERSION : RELATION -> OBJET-RELATION ...

```
INSERT INTO Polygons
    SELECT pp.name,
        CAST(
            MULTISET(    SELECT x, y
                         FROM PolyFlat   qq
                         WHERE qq.name = pp.name )
            AS PolygonType
        )
    FROM PolyFlat pp
    WHERE NOT EXISTS(
        SELECT *
        FROM PolyFlat rr
        WHERE rr.name = pp.name
        AND ( rr.x < pp.x OR rr.x = pp.x )
        AND rr.y < pp.y);
```

**Idée : ne faire l 'insertion que pour un point suivant l 'ordre lexicographic*

2. Large Objects (LOBs)

- **Oracle distingue 3types de LOBs :**
  - **BLOBs** (Binary Large Objects) & **CLOBs** (Character Large Objects) :
    - taille < 4 Go
    - stocké dans la BD mais à l 'extérieur de la relation
  - **BFILE** (Binary File)
    - taille illimitée
    - stocké à l 'extérieru de la BD (fichier référencé dans la BD)
    - NB : la cohérence des données n 'est pas assurée par Oracle
- **Fonctions de manipulation accessibles via PL/SQL :**
  - read, write, getlength, substr, instr, append, erase, compare, copy

2. Large Objects (LOBs) ...

- **Exemples :**

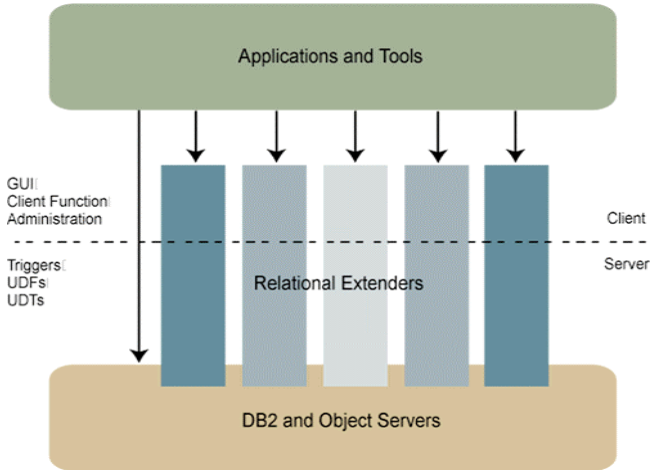
```
CREATE TABLE Personne (
    nom VARCHAR(10)
    photo    BFILE,
    cv       CLOB,
    prog     BLOB);

INSERT INTO Personne VALUES (
    ' Tintin ',
    bfilename('$envir/images ', ' tintin.jpeg),
     ' Tintin est un journaliste très connu,…… ',
    empty_blob()
);
```

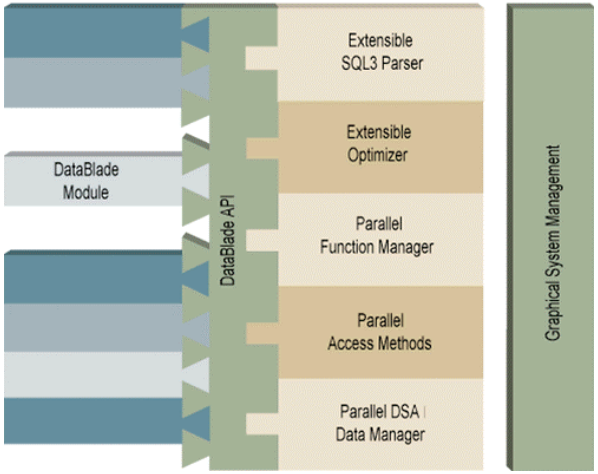

CONCLUSION

- **Etendre un système relationnel signifie ajouter :**
  - des ADTS ou des LOBs :
    - UDTs (User-defined Data Types)
    - UDFs (User-defined Functions) ou procédures stockées
  - des méthodes d 'accès pour les UDTs
  - adaptation de l 'optimiseur :
    - statistiques, propriétés des opérateurs, règles de transformation
  - des triggers
- **Les technologies :**
  - DB2 relational extenders
  - Informix DataBlades
  - Oracle Cartridges

3. DB2 Relational Extenders



3. Informix DataBlades



3. Oracle Cartridges