Multimedia Database Systems: casi d'uso

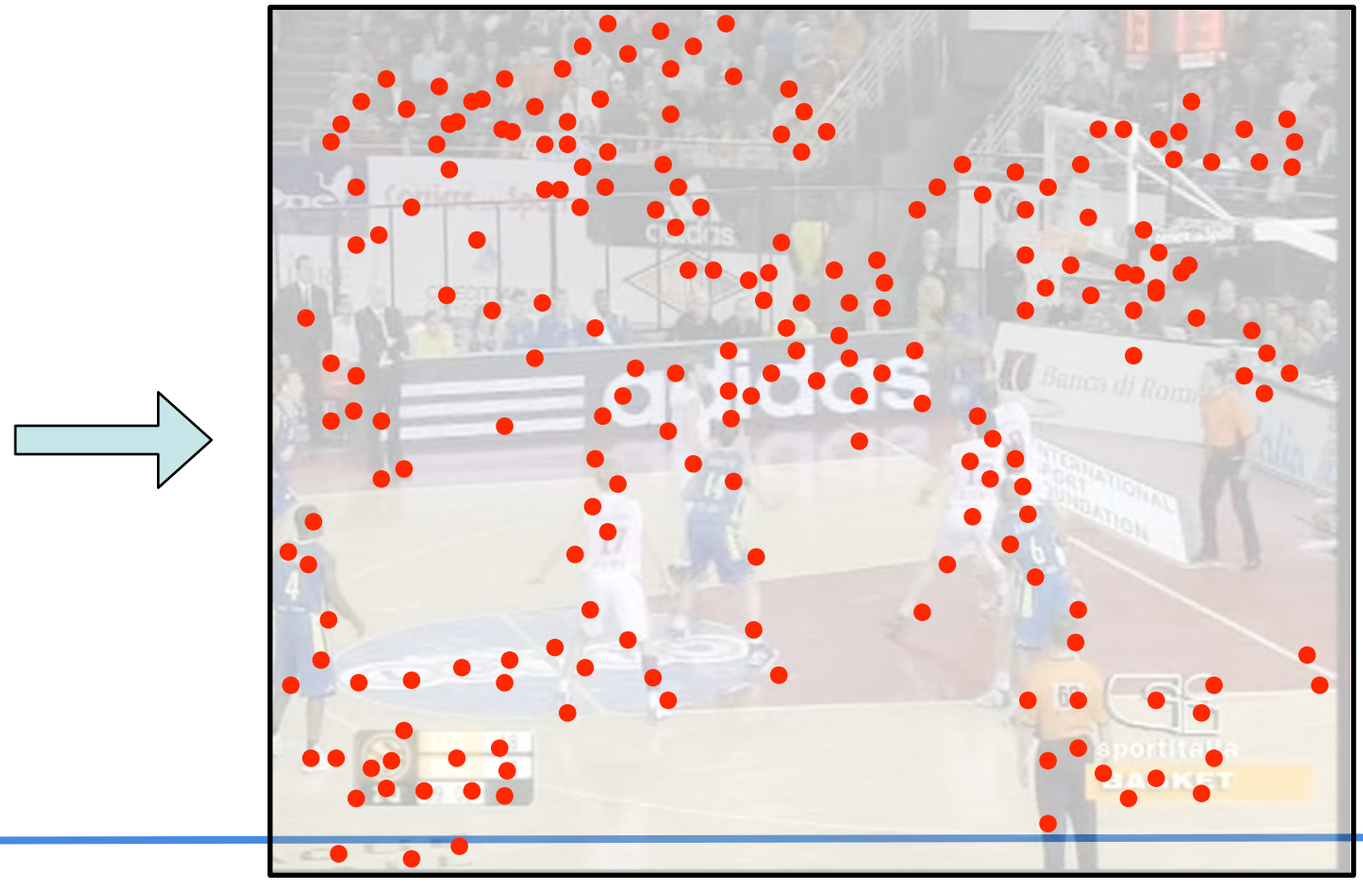
- Un Multimedia Database Management System (MMDBMS) deve fornire supporto a tipi di dati multimediali oltre che le tradizionali funzioni DBMS
- MMDBMS commerciali sono ancora poco popolari e spesso poco supportati
  - es. IBM DB2 AIV extenders non sono più supportati dalla versione DB2 9.1

Database relazionali

- In alcuni casi non è necessario usare un MMDBMS
- Es.: quando non è necessario usare un indice basato su dati multimediali
- Caso d'uso: sistema di riconoscimento di trademark in video sportivi
- Obiettivo: determinare gli istanti temporali in cui appare un marchio
- Soluzione: memorizzare su DBMS l'istante di apparizione (e la posizione) dei marchi

Caso d'uso: riconoscimento di marchi in video sportivi

- Per determinare la presenza di un marchio si analizzano i frame del video e se ne estraggono gli interest points (es. SIFT).
- I dati relativi ai punti (vettori ad alta dimensionalità) si salvano su DBMS relazionale (es. MySQL)



- L'operazione di matching dei marchi richiede l'analisi di tutti i frame del video
  - l'indice è creato sul timecode

	Campo	Tipo	Collation	Attributi	Null	Predefinito	Extra	Azione
☐	id	int(11)			No		auto_increment	
☐	timecode	int(11)			Sì	*NULL*		
☐	scale	float			Sì	*NULL*		

Indici:

Nome chiave	Tipo	Cardinalità	Azione	Campo
PRIMARY	PRIMARY	2734302		id
timecode	INDEX	6904		timecode

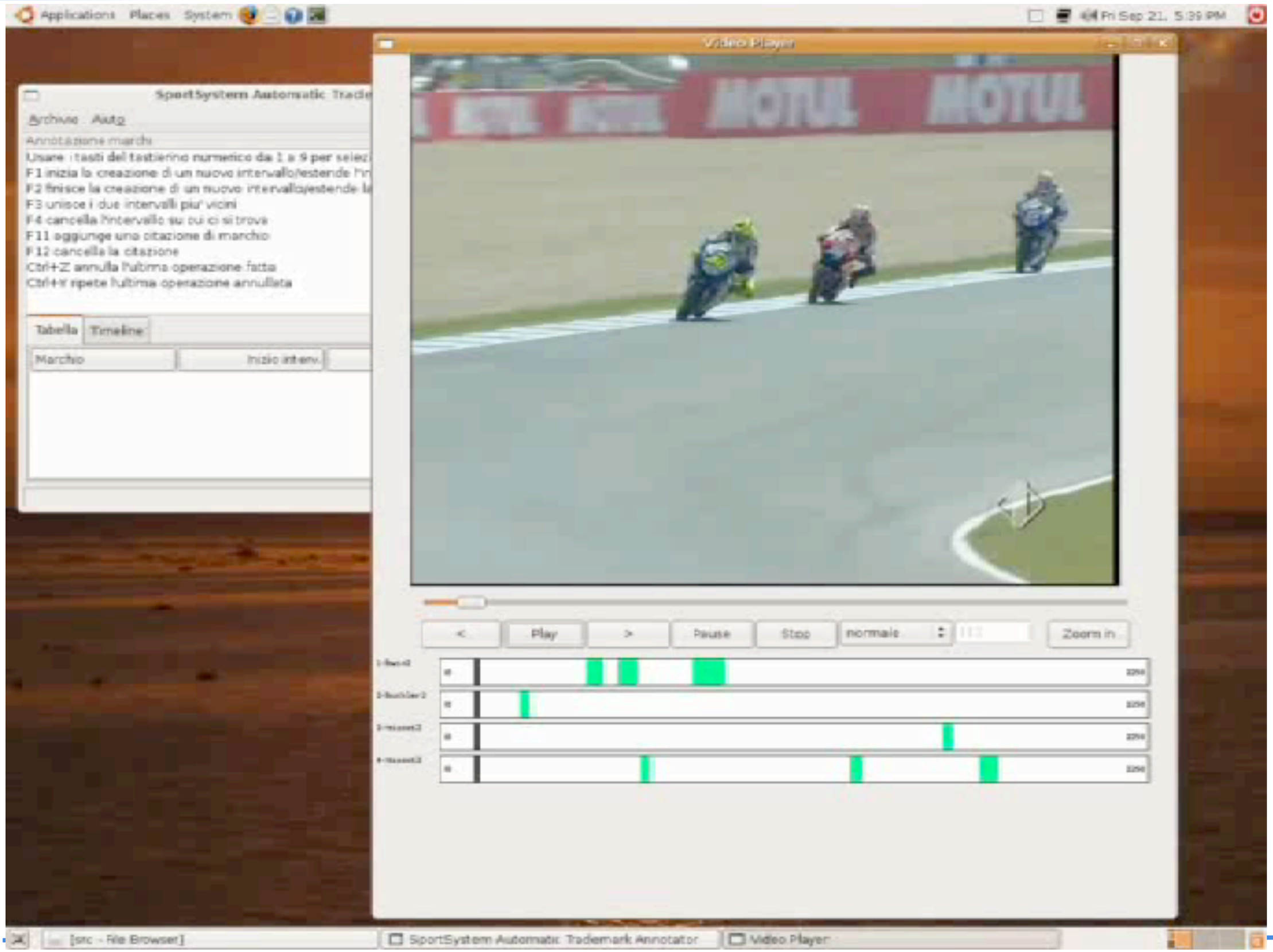
Crea un indice su 1 columns Esegui

Spazio utilizzato

Tipo	Utilizzo	
Dati	1.442	MiB
Indice	57.606	KiB
Totale	1.498	MiB

- I risultati del riconoscimento di un marchio (tempo e posizione) sono memorizzabili in una semplice tabella...

Trademark SIFT keypoints


Applications Places System
Fri Sep 21, 5:39 PM
SportSystem Automatic Tracke
Archivio Aiuto
Annotazione marchi
Usare i tasti del tastierino numerico da 1 a 9 per selez
F1 inizia la creazione di un nuovo intervallo/estende l'in
F2 finisce la creazione di un nuovo intervallo/estende la
F3 unisce i due intervalli piu' vicini
F4 cancella l'intervallo su cui ci si trova
F11 aggiunge una citazione di marchio
F12 cancella la citazione
Ctrl+Z annulla l'ultima operazione fatta
Ctrl+Y ripete l'ultima operazione annullata
Tabella
Timeline
Marchio
Inizio interv.
Video Player
MOTUL
MOTUL
<
Play
>
Pause
Stop
normale
Zoom in
[src - File Browser]
SportSystem Automatic Trademark Annotator
Video Player

Database relazionali e XML

- I metadati relativi a oggetti multimediali possono essere comodamente rappresentati usando XML:
  - i dati sono facilmente interscambiabili (es. in ambiente distribuito)
  - struttura gerarchica che si adatta bene a molti tipi di dati multimediali (es. le possibili suddivisioni di un video)
  - consente di rappresentare facilmente strutture di dati come alberi, liste e record

- Caso d'uso: sistema di annotazione di video sportivi ASSAVID
- Obiettivo: annotare a diversi livelli sintattici e semantici video sportivi, in ambiente distribuito
- Soluzione: definizione di un formato XML comune per l'annotazione dei video. Mappatura della fusione dei dati XML su DBMS (PostgreSQL). Le tabelle del DBMS sono create dal programma Java responsabile della fusione dei dati.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT document (summary?, annotation*)>
<!ELEMENT summary (#PCDATA)>
<!ELEMENT annotation (timecode, category, value*)>
<!ELEMENT timecode EMPTY>
<!ELEMENT category EMPTY>
<!ELEMENT value (#PCDATA)>
<!ATTLIST document
name CDATA #REQUIRED
date CDATA #IMPLIED
version CDATA #IMPLIED
author CDATA #REQUIRED
producer CDATA #REQUIRED
producerIp CDATA #IMPLIED
>
<!ATTLIST timecode
time CDATA #IMPLIED
frames CDATA #IMPLIED
>
<!ATTLIST category
name (Cue | Description | Sport | Image | Audio | Shot.Change | Text | Rating) #REQUIRED
subname (Event | Personality | New.Speaker | Music | Sounds | Caption | On.Screen) #IMPLIED
>
<!ATTLIST value
kind (string | image | time | frames | integer | float) #IMPLIED
>
```

il DTD usato in ASSAVID

vista delle tabelle

TimeCode	Cue	Description	Sport	Audio	Shot.Change
22:37:30.00			CloseUp : 0.917053		
22:37:30.10	three_quarters				
22:37:30.20	three_quarters				
22:37:31.05	three_quarters				
22:37:31.15	three_quarters				
22:37:31.20					1
22:37:31.22	midfield	Mosaic Duration: 80 msec			
22:37:31.23					1
22:37:32.00	right_midfield	Mosaic Duration: 1800 msec	CloseUp : 0.926974	WhistleReferee : 0.857445	

22:43:05.21		Soccer highlight: Forward pass	CloseUp : 0.967876
22:43:06.00	right_midfield		
22:43:06.00		Great Pass	
22:43:06.09		Good pass by	
22:43:06.10	right_midfield		

DB relazionali per XML

- Alcuni DBMS sono in grado di importare ed esportare XML in modo nativo
  - es.: IBM DB2, Oracle, MS SQL, MySQL, Oracle Berkeley DB XML
  - in importazione i documenti XML possono essere rimappati in tabelle, o inseriti in una colonna

Colonne XML

- è bene usare memorizzare i documenti XML in colonne (VARCHAR o CLOB) se:
  - si vogliono archiviare interi documenti XML
  - si aggiornano poco frequentemente i dati XML
  - si leggono molto frequentemente i dati XML
- tipicamente si accede ai nodi XML con XPath
- le possibilità di indicizzazione dipendono dal DB usato
  - es. DB2 side tables

Colonne XML: esempio MySQL

- In MySQL 5.1.5 c'è un supporto iniziale all'importazione di XML.
- Esempio di tabella con colonna che contiene dati XML:

```
CREATE TABLE x (doc VARCHAR(150));

INSERT INTO x VALUES

('

<book>

<title>A guide to the SQL standard</title>

<author>

<initial>CJ</initial>

<surname>Date</surname>

</author>

</book>

');

INSERT INTO x VALUES

('

<book>

<title>SQL:1999</title>

<author>

<initial>J</initial>

<surname>Melton</surname>

</author>

</book>

');
```

- Per accedere/modificare i nodi si usano Extractvalue / UpdateXML:

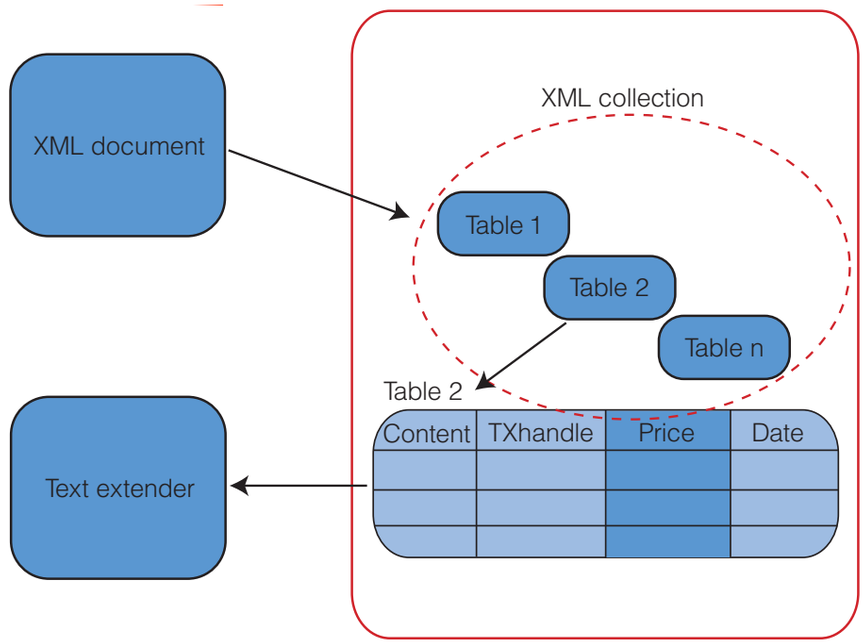
```
mysql> SELECT EXTRACTVALUE(doc,'/*/*/initial') FROM x;
+---------------------------------+
| EXTRACTVALUE(doc,'/*/*/initial') |
+---------------------------------+
| CJ |
| J |
+---------------------------------+
2 rows in set (0.01 sec)
```

```
mysql> SELECT extractValue(doc,'/book/child::*') FROM x;
+-------------------------------------------+
| extractValue(doc,'/book/child::*') |
+-------------------------------------------+
| A guide to the SQL standard |
| SQL:1999 |
+-------------------------------------------+
2 rows in set (0.00 sec)
```

```
mysql> select UpdateXML(doc,'/book/author/initial','!!') from x;
+-----------------------------------------------------------+
| UpdateXML(doc,'/book/author/initial','!!') |
+-----------------------------------------------------------+
|
<book>
<title>A guide to the SQL standard</title>
<author>
!!
<surname>Date</surname>
</author>
</book> |
|
<book>
<title>SQL:1999</title>
<author>
!!
<surname>Melton</surname>
</author>
</book> |
+-----------------------------------------------------------+
2 rows in set (0.00 sec)
```

Collezioni XML

- L'alternativa alla memorizzazione in una colonna è la rimappatura di un documento XML in tabelle di un DB relazionale, ed è bene usarla se:
  - i dati si aggiornano frequentemente
  - si vogliono salvare i dati
  - si vogliono comporre documenti XML a partire dalla tabelle



- In DB2 si deve definire un Document Access Definition (DAD) file che specifica come rimappare attributi ed elementi sulle tabelle
- il processo è anche automatizzabile:

DTD

```
<!ELEMENT Order (OrderNum, Date, CustNum, Item*)>
<!ELEMENT OrderNum (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT CustNum (#PCDATA)>

<!ELEMENT Item (ItemNum, Quantity, Part)>
<!ELEMENT ItemNum (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>

<!ELEMENT Part (PartNum, Price)>
<!ELEMENT PartNum (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
```

Tables

- In DB2 si deve definire un Document Access Definition (DAD) file che specifica come rimappare attributi ed elementi sulle tabelle
- il processo è anche automatizzabile:

```
                    DTD                                          Tables
==================================================      ================

<!ELEMENT Order (OrderNum, Date, CustNum, Item*)>       ==>     Table Order
<!ELEMENT OrderNum (#PCDATA)>                                      Column OrderPK
<!ELEMENT Date (#PCDATA)>
<!ELEMENT CustNum (#PCDATA)>

<!ELEMENT Item (ItemNum, Quantity, Part)>               ==>     Table Item
<!ELEMENT ItemNum (#PCDATA)>                                       Column ItemPK
<!ELEMENT Quantity (#PCDATA)>

<!ELEMENT Part (PartNum, Price)>                        ==>     Table Part
<!ELEMENT PartNum (#PCDATA)>                                       Column PartPK
<!ELEMENT Price (#PCDATA)>
```

- In DB2 si deve definire un Document Access Definition (DAD) file che specifica come rimappare attributi ed elementi sulle tabelle
- il processo è anche automatizzabile:

DTD		Tables
`<!ELEMENT Order (OrderNum, Date, CustNum, Item*)>`	==>	Table Order
`<!ELEMENT OrderNum (#PCDATA)>`		Column OrderPK
`<!ELEMENT Date (#PCDATA)>`		**Column OrderNum**
`<!ELEMENT CustNum (#PCDATA)>`		**Column Date**
		Column CustNum
`<!ELEMENT Item (ItemNum, Quantity, Part)>`	==>	Table Item
`<!ELEMENT ItemNum (#PCDATA)>`		Column ItemPK
`<!ELEMENT Quantity (#PCDATA)>`		**Column ItemNum**
		Column Quantity
`<!ELEMENT Part (PartNum, Price)>`	==>	Table Part
`<!ELEMENT PartNum (#PCDATA)>`		Column PartPK
`<!ELEMENT Price (#PCDATA)>`		**Column PartNum**
		Column Price

- In DB2 si deve definire un Document Access Definition (DAD) file che specifica come rimappare attributi ed elementi sulle tabelle

- il processo è anche automatizzabile:

```
                      DTD                                                 Tables
=================================================                  =================

<!ELEMENT Order (OrderNum, Date, CustNum, Item*)>      ==>         Table Order
<!ELEMENT OrderNum (#PCDATA)>                                         Column OrderPK
<!ELEMENT Date (#PCDATA)>                                             Column OrderNum
<!ELEMENT CustNum (#PCDATA)>                                          Column Date
                                                                      Column CustNum

<!ELEMENT Item (ItemNum, Quantity, Part)>              ==>         Table Item
<!ELEMENT ItemNum (#PCDATA)>                                          Column ItemPK
<!ELEMENT Quantity (#PCDATA)>                                         Column ItemNum
                                                                      Column Quantity
                                                                      Column OrderFK

<!ELEMENT Part (PartNum, Price)>                       ==>         Table Part
<!ELEMENT PartNum (#PCDATA)>                                          Column PartPK
<!ELEMENT Price (#PCDATA)>                                            Column PartNum
                                                                      Column Price
                                                                      Column ItemFK
```

DB XML nativi

- Definiscono il documento XML come unità (logica) fondamentale di memorizzazione, come i DB relazionali usano una tupla
- Non definiscono necessariamente nessun modello di storage fisico

DB XML nativi: text based

- Conservano l'XML come testo, per es. sotto forma di:
  - BLOB/CLOB
  - file system
  - formato proprietario
- usano indici specifici per raggiungere le varie parti del XML

DB XML nativi: PDOMs

- I DB basati su persistent DOM implementano il DOM su uno storage persistente:
  - il DOM è live, per cui i cambiamenti si riflettono nel DB, a differenza di altri approcci in cui il DOM modificato deve essere esplicitamente aggiornato nello storage
  - hanno il ruolo dei OODBMS per i linguaggi ad oggetti...

DB XML nativi: model-based

- si memorizza il modello del documento in uno DB relazionale o ad oggetti
- la performance, specie nel retrieval, dipende dal DB sottostante
- non sono molti, vedi:

  XML Database Products
  http://www.rpbourret.com/xml/XMLDatabaseProds.htm

- DB2 e Oracle hanno funzioni per la gestione di XML nativo, in aggiunta alle funzioni di rimappatura viste prima
- in generale i DB XML non sono ancora particolarmente maturi
  - un problema è lo standard XQuery che non consente l'update (in pratica è solo l'equivalente di SELECT); XQuery Update è attualmente working draft

Object-Relational DB

- Sono DB che uniscono le caratteristiche dei DB relazionali con quelli ad oggetti.
- PostgreSQL è un DB object-relational
- si sfrutta le funzioni object oriented per definire tipi di dati ed operazioni

- Per es. si può rappresentare un histo di colore come un vettore

```
CREATE TABLE image (
        name            varchar(30),
        larghezza       integer,
        altezza                 integer,
        vettore         float[]
);
```

- L'operazione di comparazione tra histo (per trovare immagini simili) può essere scritta come funzione C/C++ richiamabile da SQL, o anche in PL/pgSQL

```
CREATE FUNCTION name(…) RETURNS …
AS '…'
LANGUAGE '…';
```

Tipo di ritorno

Corpo della funzione o nome assoluto della libreria dinamica.

Tipi dei parametri passati alla Funzione, separati da virgola

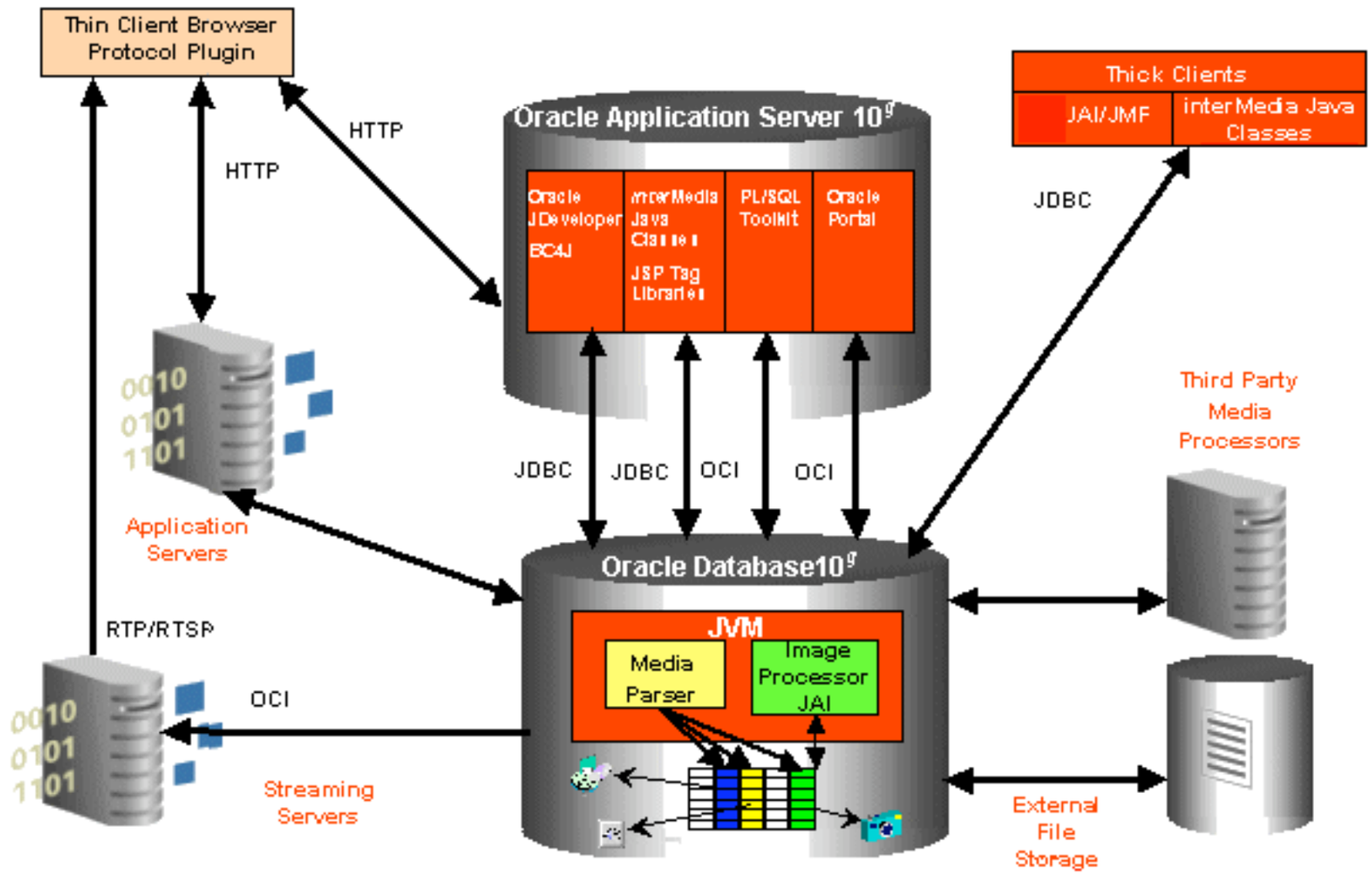
Linguaggio utilizzato (SQL, C, …)

- Si deve tenere conto che PL/pgSQL non è molto indicato per lavorare su matrici e vettori tipicamente usati per rappresentare dati multimediali
- meglio usare C/C++
- la funzione può essere usata nelle SELECT

```
/* Calcola la distanza fra due immagini espresse tramite i loro istogrammi
 * di colore.
 */
float64 distance(ArrayType *record, ArrayType *query) {

        int i, j, h,v;
        float64 ret, appo; /* Valore di ritorno e di appoggio. */
        float64 diff, start_diff; /* Differenza fra i vettori. */
        float64 temp, start_temp; /* Vettore di appoggio. */

        /* Numero di elementi degli array passati. */
        int4 nitems_record = (int4) getNitems(ARR_NDIM(record), ARR_DIMS(record));
        int4 nitems_query = (int4) getNitems(ARR_NDIM(query), ARR_DIMS(query));

        /* Restituisce il puntatore all'area dati di ArrayType e li casta a float64. */
        float64 data_record = (float64) ARR_DATA_PTR(record);
        float64 data_query = (float64) ARR_DATA_PTR(query);

        /* Alloco un float64 per il valore di ritorno e un di appoggio. */
        ret = (float64) palloc(sizeof(float64data));
        appo = (float64) palloc(sizeof(float64data));

        /* Alloco un float64 per la differenza fra i due vettori. */
        diff = (float64) palloc(sizeof(float64data)*NUM_COLOR);
        start_diff = diff; /* Salvo il punto iniziale. */

        /* Alloco un float64 per il vettore di appoggio. */
        temp = (float64) palloc(sizeof(float64data)*NUM_COLOR);
        start_temp = temp; /* Salvo il punto iniziale. */

        /* Faccio alcuni controlli. */
        if ((nitems_record!=TOTELEMENTI)||(nitems_query!=TOTELEMENTI)) {
                return NULL;
        }

        *ret = 0;
        for (v=0 ; v<VLENGTH ; v++) {
                for (h=0 ; h<HLENGTH ; h++) {

                        /* Ripristino i puntatori */
                        diff = start_diff;
                        temp = start_temp;

                        /* Calcolo la differenza fra i due vettori. */
                        ...

                        *ret += sqrt(*appo)*pesi[v][h];
                }
        }

        return ret;
}
```

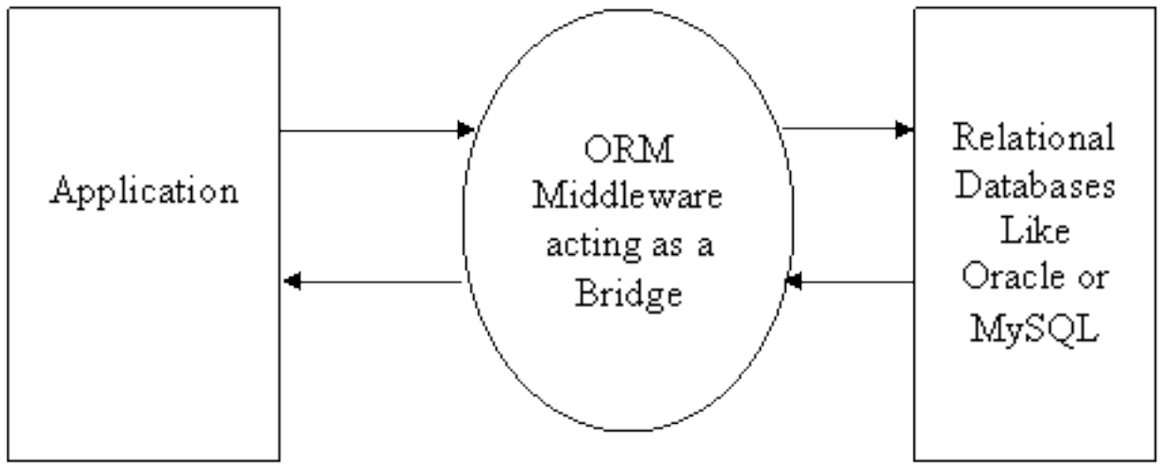
Oracle interMedia

- è una estensione object-relational specializzata per il multimedia
- supporta lo StillImage standard di SQL/MM
- crea "signature" basate su colore, texture, forma per immagini

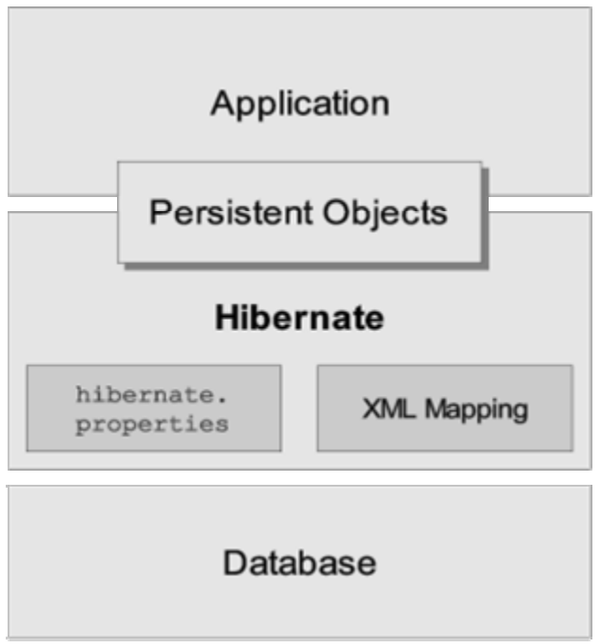


Object-relational mapping

- è una tecnica di programmazione per convertire dati tra sistemi di tipi incompatibili come linguaggi object-oriented a database
  - in pratica si crea un database a oggetti "virtuale"
  - il problema è riuscire a convertire in modo trasparente gli oggetti in un formato accettabile per il DB, ma conservando tutte le relazioni e le proprietà



- Hibernate è uno dei più famosi ORM per Java
- Hibernate si pone tra il modello ad oggetti che si ricava dalla logica della nostra applicazione e l'insieme delle tabelle, chiavi primarie e vincoli relazionali.
- Uno o più file di configurazione stabiliscono la corrispondenza tra gli oggetti della nostra applicazione e le tabelle del db.



Hibernate: classi persistenti

- Si deve seguire un modello di programmazione Plain Old Java Objects (POJO)

```
public class Cat {
private String id;
private String name;
private char sex;
private float weight;
public Cat() {
}
public String getId() {
return id;
}
private void setId(String id) {
this.id = id;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public char getSex() {
return sex;
}
public void setSex(char sex) {
this.sex = sex;
}
public float getWeight() {
return weight;
}
public void setWeight(float weight)
{
this.weight = weight;
 }
}
```

Hibernate: mapping

- Si definisce in un file XML il mappaggio tra classe e relazioni
- Hibernate può anche genereare l'SQL necessario per la creazione della tabella

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
<class name="net.sf.hibernate.examples.quickstart.Cat" table="CAT">
<!-- A 32 hex character is our surrogate key. It's automatically
generated by Hibernate with the UUID pattern. -->
<id name="id" type="string" unsaved-value="null" >
<column name="CAT_ID" sql-type="char(32)" not-null="true"/>
<generator class="uuid.hex"/>
</id>
<!-- A cat has to have a name, but it shouldn' be too long. -->
<property name="name">
<column name="NAME" length="16" not-null="true"/>
</property>
<property name="sex"/>
<property name="weight"/>
</class>
</hibernate-mapping>
```

Persistenza RDF

- Jena è una libreria per parsing, serializzazione e processamento di modelli RDF
- offre anche persistenza su diversi tipi di DB relazionali
- Jena supporta anche il linguagio di query RDF SPARQL

```
// load the the driver class
Class.forName(M_DBDRIVER_CLASS);
// create a database connection
IDBConnection conn = new DBConnection(M_DB_URL, M_DB_USER, M_DB_PASSWD, M_DB);
// create a model maker with the given connection parameters
ModelMaker maker = ModelFactory.createModelRDBMaker(conn);

// create a default model
Model defModel = maker.createDefaultModel();
...
// Open existing default model
Model defModel = maker.openModel();

// or create a named model
Model nmModel = maker.createModel("MyNamedModel");
...
// or open a previously created named model
Model prvModel = maker.openModel("AnExistingModel");
```

SPARQL

```
PREFIX abc: <http://mynamespace.com/exampleOntologie#>
SELECT ?capital ?country
WHERE {
  ?x abc:cityname ?capital.
  ?y abc:countryname ?country.
  ?x abc:isCapitalOf ?y.
  ?y abc:isInContinent abc:africa.
}
```

Compiti

- Leggere “Multimedia Database Systems: where are we now?”, H. Kosch, M. Doller, IASTED-DBA Conference, Feb. 2005
  http://www-itec.uni-klu.ac.at/~harald/

Riferimenti

- A DB2 UDB still image extender
  IBM DeveloperWorks
  http://www.ibm.com/developerworks

- Jena2 Database Interface
  http://jena.sourceforge.net/DB/

- Oracle InterMedia
  http://www.oracle.com/technology/documentation/intermedia.html

- The GiST indexing project
  http://gist.cs.berkeley.edu/