



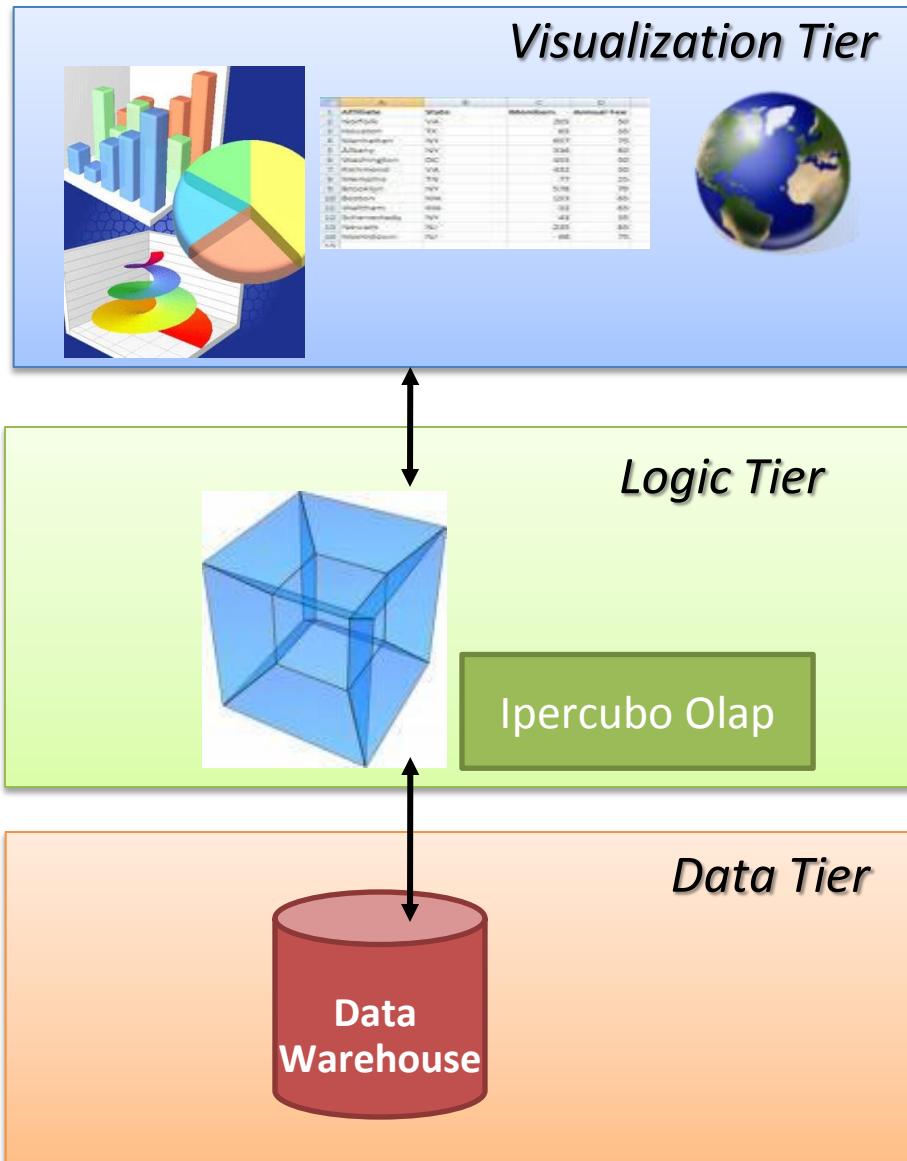
# UN' APPLICAZIONE O.L.A.P. CON MONDRIAN E JPIVOT

Basi di Dati II mod. B

A.A. 2009/2010

Vincenza Anna Leano

# ARCHITETTURA O.L.A.P.



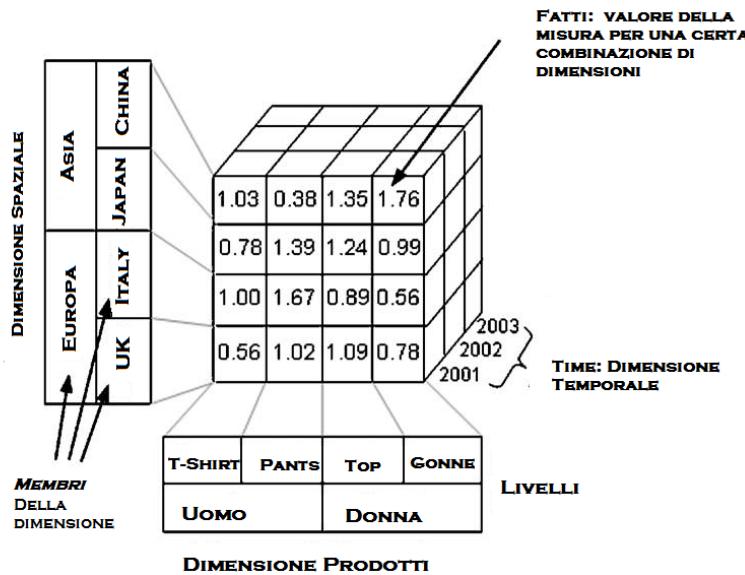
# APPLICAZIONE OLAP: How To

- Per sviluppare un'applicazione OLAP bisogna:
  - Definire un DB Relazionale che ci permetta di simulare l'ipercubo OLAP
  - Installare un Server OLAP che si occupi del paradigma multidimensionale e farlo comunicare con tale DB
  - Offrire opportuni supporti per la visualizzazione dei dati e per l'invocazione delle operazioni OLAP





# IPERCUBO OLAP



- **Ipercubo:** Un insieme di misure aggregate in base a un insieme di dimensioni
- **Fatto:** univocamente determinato da una misura ed N dimensioni.
- **Dimensioni:** rappresentano le tematiche di interesse dell'utente, considerate come le variabili indipendenti. Sono organizzate gerarchicamente in **livelli**.
- **Misure:** attributi numerici analizzati con un insieme di dimensioni, variabili dipendenti.

## ESEMPIO INQUINANTI (POLLUTANTS)

- Il nostro database *pollution* contiene i dati sulla quantità e il tipo di inquinanti misurati in determinate città d'Italia, la misurazione viene effettuata in vari periodi di tempo.
- **Fatto:** valore di un inquinante in una città in un determinato istante di tempo.
- **Misura:** Valore inquinante
- **Dimensioni:**
  - Inquinante: tipo
  - Località: regione, provincia, città
  - Tempo: anno, mese, giorno



## DIMENSION TABLE

- Le Dimension table rispondono alla parte "per (by)" di una domanda (Es: visualizza le vendite PER località)
- Devono possedere una chiave primaria costituita da un singolo campo.
- I campi della tabella, chiamati attributi, contengono la completa descrizione del record della dimensione.
- Gli attributi non possono riferirsi a campi di altre tabelle



# DIMENSION TABLE (1/2)

## Dimensione Location

- Livelli gerarchici:
  - Città (city)
  - Provincia (department)
  - Regione (region)

Location			
<u>pk_city</u>	name_city	Name_department	Name_region
...	...	...	...
...	...	...	...

## Dimensione Pollutants

- Livelli gerarchici:
  - Tipo inquinante (type)
  - Inquinante (pollutant)

Pollutants		
<u>pk_pollutant</u>	Name_pollutant	Name_type
...	...	...
...	...	...



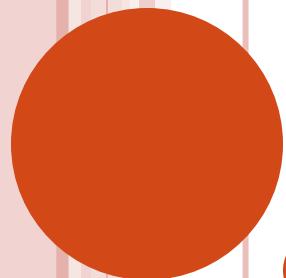


## FACT TABLE

- E' la tabella primaria del modello relazionale, memorizza i fatti dell'iper cubo.
- Una riga di questa tabella corrisponde a una misura.
- È collegata alle dimensioni tramite chiavi esterne alle dimensional table.
- La chiave primaria di questa tabella è composta dall'insieme delle chiavi esterne che puntano alle dimension table.

pollutants			
<u>Id_day</u>	<u>Id_city</u>	<u>Id_pollutant</u>	value
...	...	...	...
...	...	...	...





## LOGIC TIER

**Server OLAP: Installazione e comunicazione con il DB sottostante.**

## MONDRIAN

- Mondrian (Pentaho) è un server OLAP scritto in JAVA. Consente di analizzare in maniera interattiva e multidimensionale un vasto insieme di dati immagazzinati in database relazionali senza scrivere una riga di SQL.
- È un package software progettato per offrire le funzionalità OLAP in un framework aperto ed estendibile, al di sopra di un database relazionale.
- Mondrian consiste in un JAR che agisce come una connessione JDBC per l'OLAP.



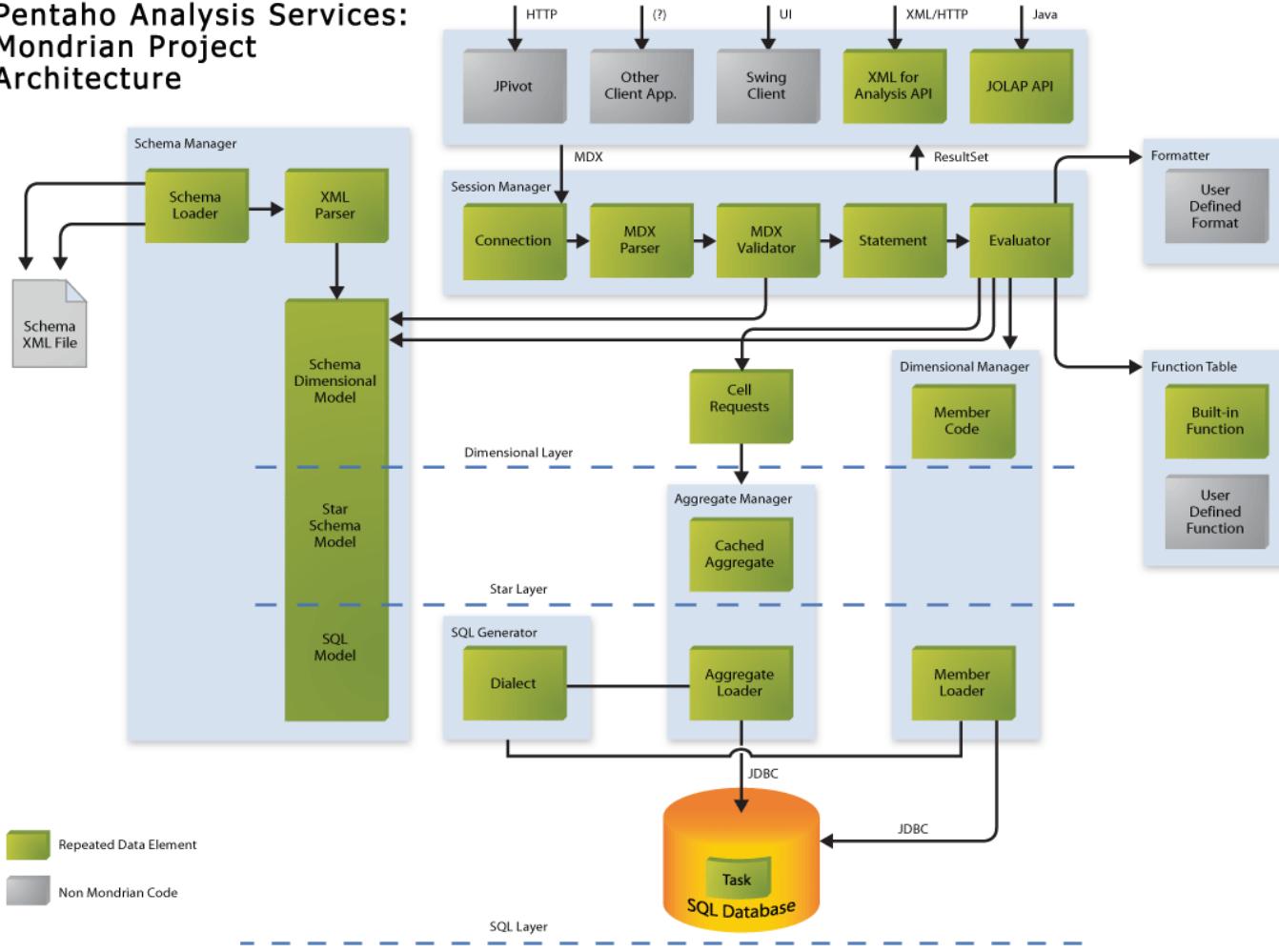
## COMPONENTI MONDRIAN

- In Mondrian l'iper cubo (Cube) su cui si opera viene definito tramite uno schema xml.
- Vengono qui definite le dimensioni con i loro livelli e le misure.
- In questo schema vengono definite le corrispondenze tra gli elementi logici del cubo e le tabelle nel DBMS.
- Mondrian trasforma questo schema in una propria struttura, su cui è possibile invocare gli operatori dimensionali tramite un apposito linguaggio: MDX



# ARCHITETTURA MONDRIAN

## Pentaho Analysis Services: Mondrian Project Architecture



## ARCHITETTURA MONDRIAN: COMPONENTI

- **Storage Layer:** è l'RDBMS e ha la responsabilità di fornire le celle di dati aggregati e i membri dalla tabella delle dimensioni.
- **Dimensional Layer:** esegue le operazioni sull'iper cubo, in particolare effettua il parsing, valida ed esegue le interrogazioni MDX.
- **Star Layer:** mantiene una cache aggregata. Un'aggregazione è un insieme di valori delle misure ("celle") in memoria, identificate da un insieme di dimensioni su colonne.
- **Presentation Layer:** determina cosa l'utente finale vede sul suo monitor e come può interagire per eseguire nuove interrogazioni.
- I componenti di Mondrian possono essere distribuiti su più macchine, l'unico vincolo è che Dimensional e Star Layer risiedano sulla stessa macchina.



## TUTORIAL: ASSUNZIONI

- Supponiamo che già siano installati i seguenti elementi necessari per il funzionamento di Mondrian:
  - JDK (almeno la 1.4.2).
  - Un IDE per Java (Eclipse in questo tutorial).
  - Tomcat
  - Un DBMS (PostgreSQL nel nostro caso)



# INSTALLAZIONE

- Scaricare dal sito

<http://sourceforge.net/projects/mondrian/>

l'ultima versione di Mondrian (quella NON embedded).

- Scompattare lo zip in una cartella a piacimento.

- Importare il war di Mondrian, che troverete sotto la cartella lib, tramite il vostro IDE.

- Per eclipse: cliccare su File->Import, selezionare Web->War file e caricare il file che troverete sotto la cartella lib della distribuzione di Mondrian.



## COMUNICAZIONE CON IL DB

- Importare il driver jdbc del vostro DBMS sotto la cartella */WebContent/WEB-INF/lib*
  - N.B.: Aggiungendola semplicemente al classpath Mondrian non la vede, bisogna copiare il file in questa cartella.
- Aprire il file *TOMCAT\_HOME/webapps/mondrian.properties* e configurare le proprietà dell'elemento *mondrian.test.connectString* per il database che abbiamo installato:

```
mondrian.test.connectString=
  Provider=mondrian;
  Jdbc=jdbc:postgresql://localhost/NOMEDB?user=LOGIN&
  password=PSWD;
  JdbcDrivers=org.postgresql.Driver;
  Catalog=/PATH/NOMESCHEMACUBO.xml;
```

## DEFINIZIONE DEL CUBO DI ANALISI

- Per definire il cubo di analisi, Mondrian definisce uno schema, utilizzando un file xml
  - elemento Catalog della stringa di connessione
- Lo Schema mappa gli elementi logici dell'iper cubo, in quelli fisici contenuti nel database.
- Uno schema Definisce un (Iper)Cubo
- Il cubo è composto da:
  - Dimensioni: organizzate gerachicamente
  - Misure: attributi numerici



# SCHEMA XML: DEFINIZIONE CUBO

```
<?xml version="1.0"?>
<Schema>
  <Cube name="NameCube">
    <Table name="FactTableName"/>
    ...
    ...
    ...
    ...
    ...
    ...
  </Cube>
</Schema>
```

- Lo Schema definisce il cubo.
- L'attributo Name di cube sarà il nome con cui identifieremo il Cubo
- Table definisce il collegamento con il DBMS: il cubo è collegato alla tabella dei fatti.



# SCHEMA XML: DEFINIZIONE DIMENSIONI

```
<?xml version="1.0"?>
<Schema>
  <Cube name="NameCube">
    <Table name="FactTableName"/>
    <Dimension name="DimensionName"
foreignkey="ForeignKeyFactTable">
      ...
      ...
      ...
      ...
      ...
      ...
      ...
      ...
      ...
    </Dimension>
  </Cube>
</Schema>
```

- In primo luogo si dà un nome alla dimensione e la si collega al DBMS con l'attributo `foreignKey`: il nome della colonna che rappresenta la dimensione nella `factTable`

# SCHEMA XML: DEFINIZIONE GERARCHIE

```
<?xml version="1.0"?>
<Schema>
  <Cube name="NameCube">
    <Table name="FactTableName"/>
    <Dimension name="DimensionName"
      foreignKey="ForeignKeyFactTable">
      <Hierarchy hasAll="">
        primaryKey="DimTable1PK">
          <Table name="DimTable1Name"/>
          ...
          ...
        </Hierarchy>
      </Dimension>
      <Dimension>... </Dimension>
      ...
    </Cube>
  </Schema>
```

- Le dimensioni possono essere organizzate gerarchicamente
- La gerarchia viene collegata al DBMS:
  - Con l'attributo primaryKey: chiave primaria della dimension table.
  - Con il tag Table: nome della dimension table.
- HasAll è un booleano, se vero viene aggiunge il livello padre di tutti gli elementi della gerarchia



# SCHEMA XML: DEFINIZIONE LIVELLI

```
<?xml version="1.0"?>
<Schema>
  <Cube name="NameCube">
    <Table name="FactTableName"/>
    <Dimension name="DimensionName"
    foreignKey="ForeignKeyFactTable">
      <Hierarchy hasAll="" primarykey="DimTable1PK">
        <Table name="DimTable1Name"/>
        <Level name="nameLevel"
        column="fieldNameDimTable1"
        uniqueMembers=""/>
      </Hierarchy>
    </Dimension>
    <Dimension>... </Dimension>
    ...
  </Cube>
</Schema>
```

- Le gerarchie sono organizzate in livelli
- Ogni livello è collegato al DBMS tramite il nome del campo nella corrispondente DimensionTable (attributo column)
- UniqueMembers: booleano per l'ottimizzazione SQL

# SCHEMA XML: DEFINIZIONE MISURE

```
<?xml version="1.0"?>
<Schema>
  <Cube name="NameCube">
    <Table name="FactTableName"/>
    <Dimension name="DimensionName"
    foreignKey="ForeignKeyFactTable">
      <Hierarchy hasAll="" primarykey="DimTable1PK">
        <Table name="DimTable1Name"/>
        <Level name="nameLevel"
        column="fieldNameDimTable1"
        uniqueMembers="" />
      </Hierarchy>
    </Dimension>
    <Dimension>... </Dimension>
    <Measure name="MeasureName"
    column="factTableClmn" aggregator=""
    visible="" />
  </Cube>
</Schema>
```

- Name: nome misura
- Column: corrispondente colonna nella fact table, per collegare la misura al DBMS
- Aggregator: [sum|count|min|max|avg|..] operatore di aggregazione da eseguire sulla misura durante il roll-up
- Visible: booleano per visualizzare o meno il valore calcolato.



# SCHEMA XML: ESEMPIO POLLUTANT (1/2)

```
<?xml version="1.0"?>
<Schema>
  <Cube name="Pollution">
    <Table name="pollution"/>
    <Dimension name="Time" foreignKey="id_day">
      <Hierarchy hasAll="true" primaryKey="pk_day">
        <Table name="timedim"/>
        <Level name="Year" column="nameyear"
uniqueMembers="true"/>
        <Level name="Month" column="namemonth"
uniqueMembers="true"/>
        <Level name="Day" column="nameday"
uniqueMembers="true"/>
      </Hierarchy>
    </Dimension>
  </Cube>
</Schema>
```



## SCHEMA XML: ESEMPIO POLLUTANT (2/2)

```
<Dimension name="Location" primaryKey="id_city">
  <Hierarchy hasAll="true" primaryKey="pk_city">
    <Table name="location"/>
    <Level name="Region" column="nameregion" uniqueMembers="true"/>
    <Level name="Department" column="namedepartment"
uniqueMembers="true"/>
    <Level name="City" column="namecity" uniqueMembers="true"/>
  </Hierarchy>
</Dimension>
<Dimension name="Pollutants" primaryKey="id_pollutant">
  <Hierarchy hasAll="true" primaryKey="pk_pollutant">
    <Table name="pollutants"/>
    <Level name="Type" column="nametype" uniqueMembers="true"/>
    <Level name="Pollutant" column="namepollutant" uniqueMembers="true"/>
  </Hierarchy>
</Dimension>
  <Measure name="PollutionValue" column="value" aggregator="avg"
visible="true"/>
</Cube>
</Schema>
```

## MDX

- L'MDX è un linguaggio standard per le interrogazioni sui database multidimensionali, proprio come SQL è il linguaggio standard per le interrogazioni su quelli relazionali.
- MDX è stato introdotto dalla Microsoft nel 1998 con Microsoft SQL Server OLAP Services.
- Ad oggi è lo standard per questo tipo di applicazioni ed è adottato da tutti i provider OLAP.



# SINTASSI MDX

```
SELECT
  { [Measures] . [Measure1Name] , ... } ON COLUMNS ,
  { [Dim1Name] . [LevelName] , ... } ON ROWS
FROM  [CubeName]
WHERE <Slicer_Condition>
```

- SELECT : Misure e Dimensioni da mostrare su righe e colonne, al livello di dettaglio desiderato
- FROM: Cubo su cui agire
- WHERE: Opzione di slicer, spesso nella forma [AxisName].[Level|Member]



## ESEMPIO MDX

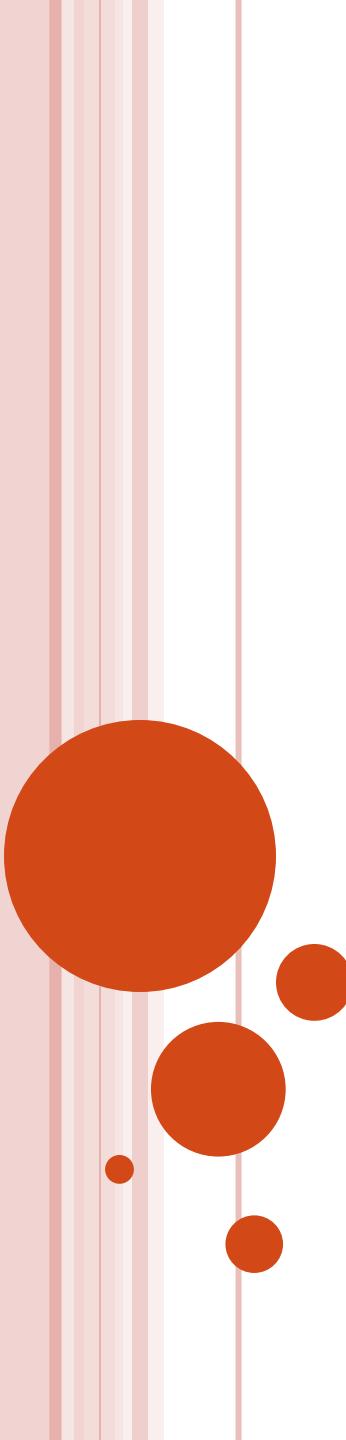
- Selezioniamo il nostro cubo Pollution attraverso la query MDX:

```
SELECT
  { [Measures].[PollutionValue] } ON COLUMNS,
  { [Location], [Pollutants], [Time] } ON ROWS
FROM Pollution
```

- Se volessimo analizzare solo l'inquinamento nel 2001 basterebbe aggiungere:

```
SELECT
  { [Measures].[PollutionValue] } ON COLUMNS,
  { [Location], [Pollutants] } ON ROWS
FROM Pollution
WHERE ([Time].[2001])
```





## VISUALIZATION TIER

JPivot

## JPIVOT

- JPivot è un package software progettato per offrire uno strato di presentazione grafico e web-based e il supporto agli operatori dimensionali.
- Offre dei tag JSP specifici per costruire facilmente interfacce grafiche atte ad esplorare il Data Warehouse.
- Svantaggi: poco flessibile e non facilmente integrabile con altri componenti, l'aggiornamento degli elementi grafici richiede un post-back completo della pagina.
- Vantaggi: Dimensional Layer, JPivot offre il parsing degli operatori multidimensionali in query MDX, e restituisce il risultato delle interrogazioni tramite strutture dati standard.

## TAG JPIVOT: JP

- **<Jp:mondrianQuery>**: Crea un attributo di sessione che consente l'accesso ad una query Mondrian, ai suoi risultati e alla navigazione.

```
<jp:mondrianQuery id= "SessionObjectID"  
    jdbcDriver=""  
    jdbcUrl=""  
    jdbcUser=""  
    jdbcPassword=""  
    catalogUri="PathXMLCubeSchema" >  
    MDX_QUERY  
</jp:mondrianQuery>
```



## TAG JPIVOT

- **<jp:table>**: Crea una componente “Pivot table”. Non produce nessun output, il componente deve essere reso visibile tramite il tag WCF render tag. Contiene in id dell’elemento, il riferimento alla query MDX che recupera I dati da immettere nella Pivot Table e un parametro che setta la visibilità dell’elemento.

```
<jp:table  
id= "SessionObjectID"  
query="#refToIdJp:MondrianQuery"  
visible=[true|false] />
```



## WCF: WEB COMPONENT FRAMEWORK

- I **Web Component Framework** (WCF) sono delle classi che aiutano a creare, presentare e validare form HTML utilizzando XML e XSLT.
- WCF incapsula i componenti riusabili sviluppati per Jpivot
- WCF è stato progettato per essere integrato facilmente con altre applicazioni web esistenti come JSF (JavaServer Faces), Struts etc.



## TAG WCF

- **<wfc:include>**: Include una determinata pagina JSP se il paramentro http corrisponde. La pagina caricata sarà data dal path: prefix + parameter + suffix.

```
<wfc:include id= "SessionObjectID"  
httpParam="paramToMatch" prefix="" suffix="" />
```

- **<wfc:render>**: I componenti WFC producono internamente documenti XML che sono trasformati utilizzando XSLT nel fromato di output tramite questo tag

```
<wfc:render ref= "#{objectToRender}"  
xslUri="pathXsl" cache="[true|false]" />
```

## RIASSUMENDO...

Per utilizzare Jpivot tramite i tag jsp occorre:

- Definire una query MDX per selezionare il Cubo (<jp:queryMondrian>)
- Creare un elemento grafico, per esempio una tabella, che lo contenga (<jp:table>)
- Rendere visibile tale contenuto attraverso i WCF(<wcf:render>)



## TUTORIAL

- Jpivot è incluso nella distribuzione di Mondrian che abbiamo scaricato.
- Per visualizzare una pivot table del nostro Ipercubo *pollution* basterà modificare il tag `jp:mondrianQuery` nel file `/Web-Inf/queries/mondrian.jsp`:
  - Inserendo gli stessi parametri di connessione di `mondrian.properties`
  - Specificando la query MDX che seleziona il cubo
- In `testpage.jsp` già sono presenti i tag `WFC` per renderizzare la tabella



# FILE MONDRIAN.JSP

```
<%@ page session="true" contentType="text/html; charset=ISO-8859-1" %>
<%@ taglib uri="http://www.tonbeller.com/jpivot" prefix="jp" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<%-- uses a dataSource --%>
<%-- jp:mondrianQuery id="query01" dataSource="jdbc/MondrianFoodmart" catalogUri="/WEB-INF/demo/FoodMart.xml" --%>

<%-- uses mysql --%>
<%-- jp:mondrianQuery id="query01" jdbcDriver="com.mysql.jdbc.Driver" jdbcUrl="jdbc:mysql://localhost/foodmart" catalogUri="WEB-INF/demo/FoodMart.xml" --%>

<%-- uses a role defined in FoodMart.xml --%>
<%-- jp:mondrianQuery role="California manager" id="query01" jdbcDriver="org.postgresql.Driver" jdbcUrl="jdbc:postgresql://localhost/foodmart" catalogUri="WEB-INF/demo/FoodMart.xml" --%>

<jp:mondrianQuery id="query01" jdbcDriver="org.postgresql.Driver" jdbcUrl="jdbc:postgresql://localhost/pollution"
jdbcUser="postgres" jdbcPassword="nekoneko" catalogUri="/WEB-INF/queries/GeoPollution.xml">
select
  {[Measures].[PollutionValue]} on columns,
  {[Location],[Pollutants], [Time]} on rows
  from Pollution
</jp:mondrianQuery>

<c:set var="title01" scope="session">Test Query uses Mondrian OLAP</c:set>
```

## BIBLIOGRAFIA

- **Mondrian Documentation:**

<http://mondrian.pentaho.org/documentation/doc.php>

- **Tutorial di G. Granatella:**

<http://www.giampierogranatella.com/blog/?p=119>

- **Jpivot:** <http://jpivot.sourceforge.net/>

