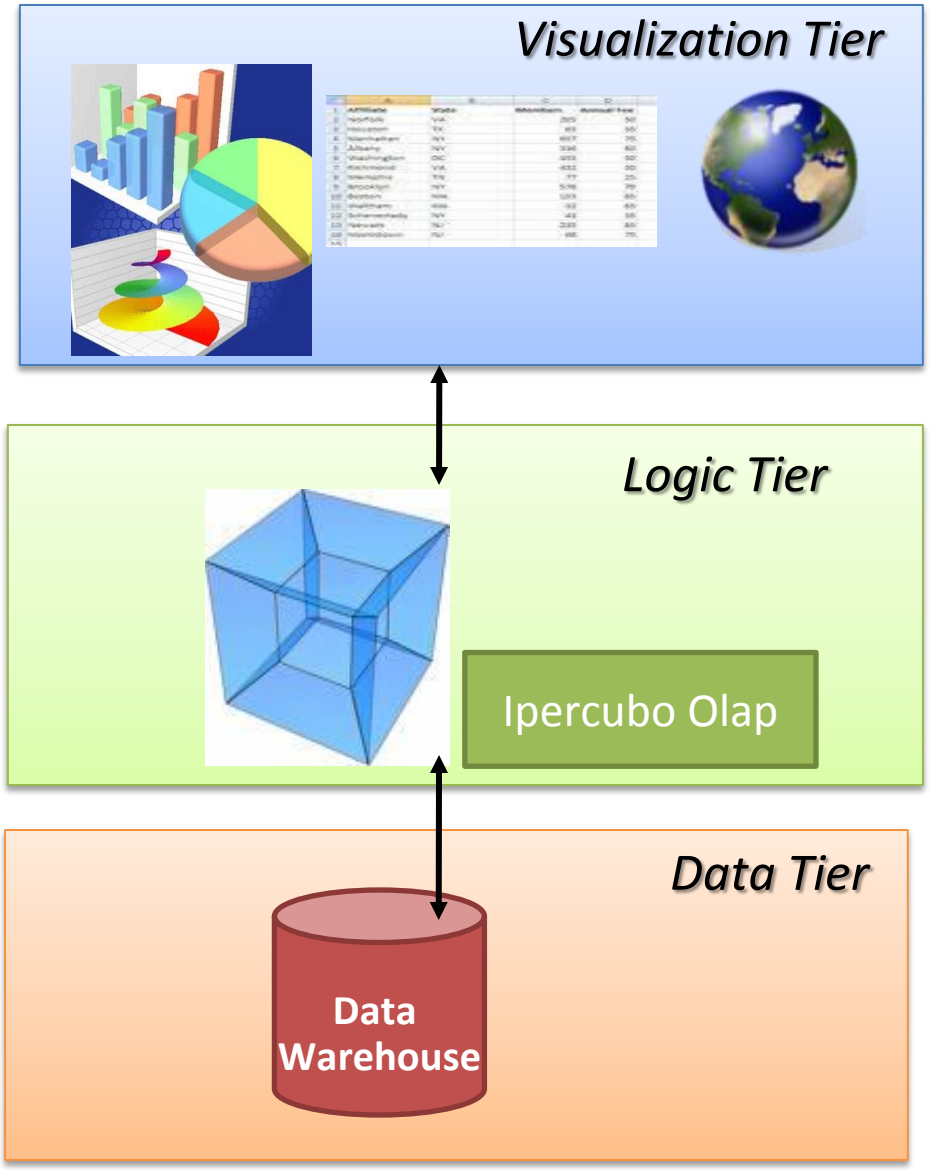
Un' Applicazione O.L.A.P. con Mondrian e JPivot

Dott.sa Vincenza Anna Leano
vincenzaanna.leano@unina.it

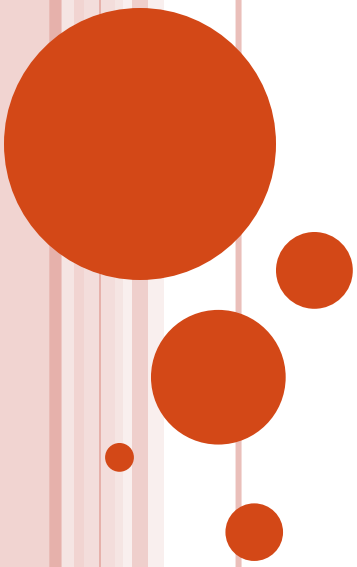
Basi di Dati II mod. B
Prof. F. Cutugno
A.A. 2010/2011

Architettura Applicazioni O.L.A.P.



Applicazione OLAP: How To

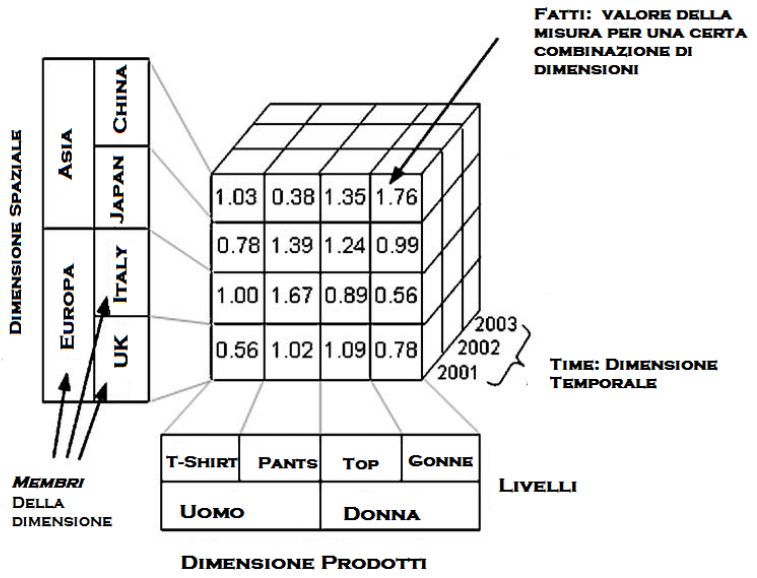
- Per sviluppare un'applicazione OLAP bisogna:
  - Definire un DB Relazionale che ci permetta di simulare l'ipercubo OLAP
  - Installare un Server OLAP che si occupi del paradigma multidimensionale e farlo comunicare con tale DB
  - Offrire opportuni supporti per la visualizzazione dei dati e per l'invocazione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



Data Tier

Organizzare il DBMS per la simulazione dell'ipercubo

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 (Polluttants)

- 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			
pk_city	name_city	Name_department	Name_region
…	…	…	…
	…	…	…

Dimensione Polluttants

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

Polluttants		
pk_polluttant	Name_polluttant	Name_type
…	…	…
	…	…

Dimension Table (2/2)

Dimensione Time

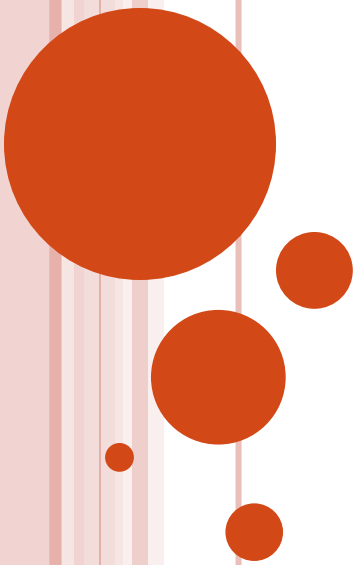
- Livelli gerarchici:
  - Anni (year)
  - Mesi (Month)
  - giorni (day)

Time			
pk_day	Name_day	Name_month	Name_year
…	…	…	…
	…	…	…

Fact Table

- E’ la tabella primaria del modello relazionale, memorizza i fatti dell’ipercubo.
- 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.

pollution			
Id_day	Id_city	Id_pollutant	value
…	…	…	…
…	…	…	…



Logic Tier

Server OLAP: Installazione e comunicazione con il DB sottostante.

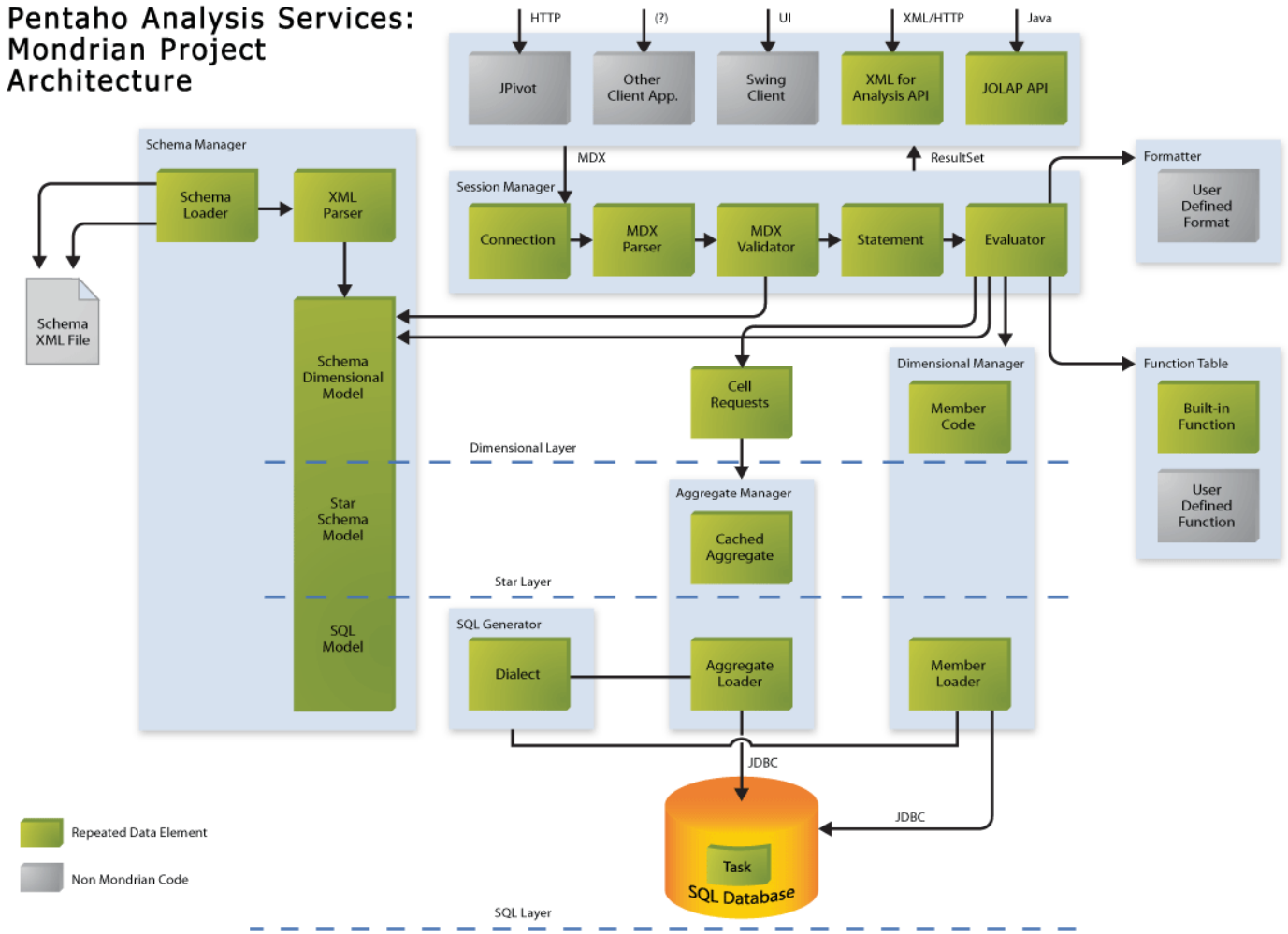
Mondrian

- 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'ipercubo (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



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'ipercubo, 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

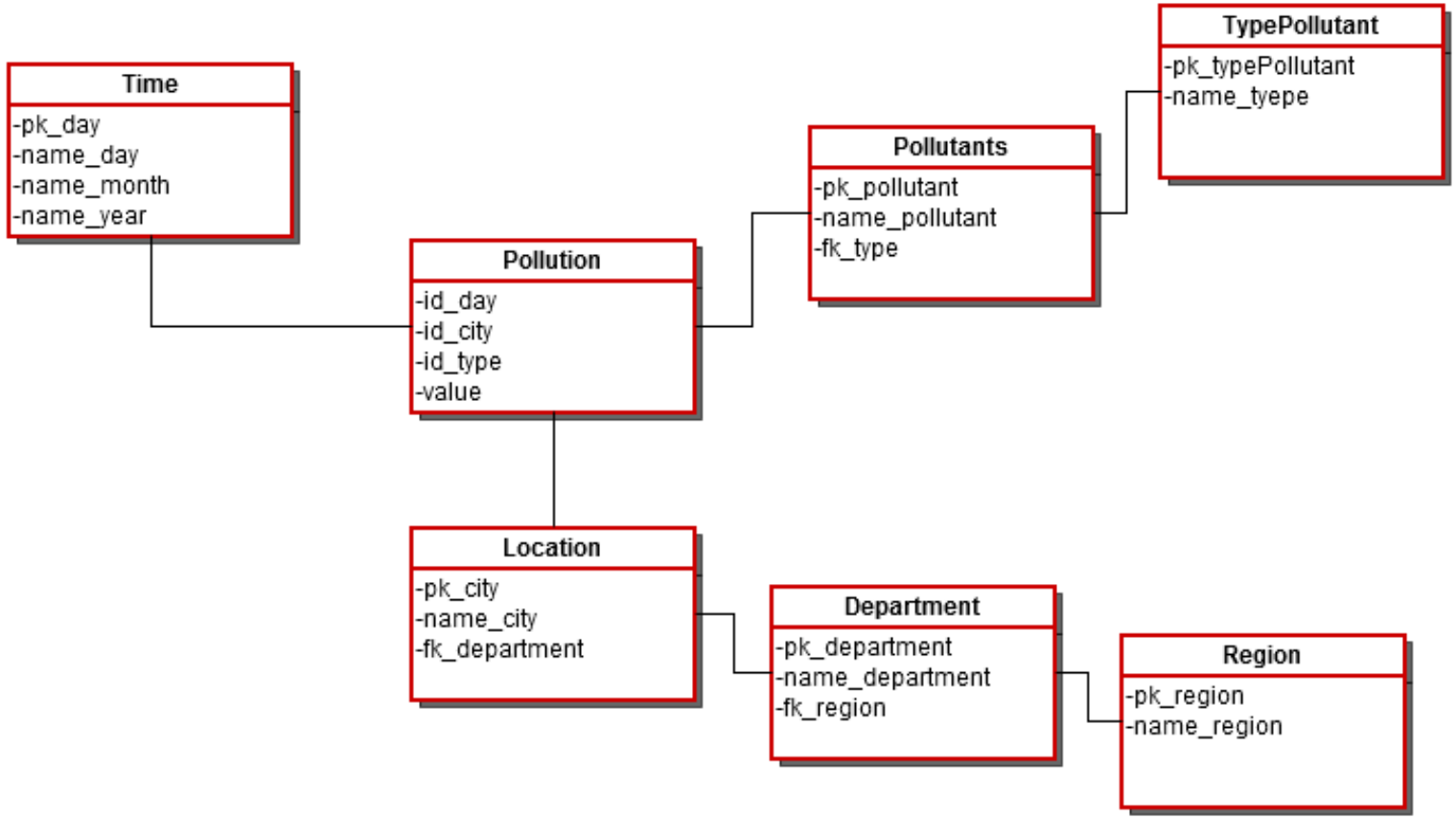
- Importare il driver jdbc del vostro DBMS sotto la cartella */WebContent/WEB-INF/lib*
  - N.B.: Aggiungendola semplicemente al classpath Mondrian non la vede, bisogna copiare il file in questa cartella.
- Aprire il file *TOMCAT_HOME*/webapps/mondrian.properties e configurare le proprietà dell'elemento *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'ipercubo, in quelli fisici contenuti nel database.
- Uno schema Definisce un (Iper)Cubo
- Il cubo è composto da:
  - Dimensioni: organizzate gerachicamente
  - Misure: attributi numerici

Diagramma Pollution



Schema

- Lo Schema definisce il cubo:
  - A schema is a collection of cubes and virtual cubes. It can also contain shared dimensions (for use by those cubes), named sets, roles, and declarations of user-defined functions.

```
<!ELEMENT Schema
  (Annotations?,(Parameter)*,(Dimension)*,(Cube)*,
  (VirtualCube)*,(NamedSet)*,(Role)*,(UserDefinedF
  unction)*)>
<!ATTLIST Schema
  name CDATA #REQUIRED
  description CDATA #IMPLIED
  measuresCaption CDATA #IMPLIED
  defaultRole CDATA #IMPLIED>
```

Cube

- Il cubo è un insieme di **dimensioni e misure.**
- Il collegamento con i valori del DBMS viene effettuato tramite l'elemento TABLE: il cubo è collegato alla tabella dei fatti.

```
<!ELEMENT Cube
   (Annotations?,(Table),(Dimension)*,(Measure)* …>
<!ATTLIST
Cubename CDATA #REQUIRED
caption CDATA #IMPLIED
description CDATA #IMPLIED
defaultMeasure CDATA #IMPLIED
cache (true|false) "true"
enabled (true|false) "true">
```

Table

- Permette il collegamento degli elementi del cubo con il DBMS sottostante
- Definisce il nome, lo schema e un eventuale alias per la tabella

```
<!ELEMENT Table (…)>
<!ATTLIST Table
name CDATA #REQUIRED
schema CDATA #IMPLIED
alias CDATA #IMPLIED>
```

Schema XML: definizione Schema e Cubo

```
<?xml version="1.0"?>
<Schema name="PollutionSchema">
  <Cube name="PollutionCube">
    <Table name="pollution"/>
                                  …


</Cube>
</Schema>
```

Dimension

- La dimensione è la variabile indipendente del cubo
- Può essere organizzata in più gerarchie
- Può essere pubblica, cioè condivisa da tutto lo schema, oppure privata, cioè di proprietà del cubo di cui è figlia.
- Collegamento al DB: foreignKey su tabella dei fatti

```
<!ELEMENT Dimension (Annotations?,(Hierarchy)*)>
<!ATTLIST Dimension
  name CDATA #REQUIRED
  type (StandardDimension|TimeDimension)#IMPLIED
  usagePrefix CDATA #IMPLIED
  foreignKey CDATA #IMPLIED
  caption CDATA #IMPLIED
  description CDATA #IMPLIED>
```

Schema XML: definizione Dimensioni

```
<?xml version="1.0"?>
<Schema name="PollutionSchema">
  <Cube name="PollutionCube">
    <Table name="pollution"/>
  <Dimension name="Time" foreignKey="id_day">
      …
  </Dimension>
…
</Cube>
</Schema>
```

Hierarchy

- La gerarchia viene collegata al DBMS:
  - primaryKey: chiave primaria della dimension table.
  - Table|Join: se abbiamo uno schema a stella o snow-flake
- hasAll è un booleano, se vero viene aggiunge il livello padre di tutti gli elementi della gerarchia

```
<!ELEMENT Hierarchy
   (Annotations?,(Table|Join;)?,(Level)*,…>
<!ATTLIST Hierarchy
name CDATA #IMPLIED
hasAll (true|false) #REQUIRED
allMemberName CDATA #IMPLIED
allMemberCaption CDATA #IMPLIED
allLevelName CDATA #IMPLIED
primaryKey CDATA #IMPLIED
primaryKeyTable CDATA #IMPLIED>
```

Schema XML: Definizione Gerarchie- Star schema

```
<?xml version="1.0"?>
<Schema name="PollutionSchema">
  <Cube name="PollutionCube">
    <Table name="pollution"/>
  <Dimension name="Time" foreignKey="id_day">
     <Table name="time"/>
      …
  </Dimension>
…
</Cube>
</Schema>
```

JOIN

- JOIN: usato per collegare le tabelle in cui è “scomposta” la dimensione
- Prende due operatori, ma è arbitrariamente innestato
- Gli operatori possono essere elementi table o elementi join

```
<!ELEMENT Join ((Table|Join|…),(Table|Join|…))>
<!ATTLIST Join
 leftAlias CDATA #IMPLIED
 leftKey CDATA #REQUIRED
 rightAlias CDATA #IMPLIED
 rightKey CDATA #REQUIRED>
```

SCHEMA XML: DEFINIZIONE GERARCHIE - SNOWFLAKE

```
<?xml version="1.0"?>
<Schema>
  …
 <Dimension name="polluttants" foreignKey="id_polluuttant">
   <Hierarchy primaryKey="pk_polluttant" primaryKeyTable=
"polluttant">
    <Join leftKey="fk_type" rightAlias="type"
     rightKey="pk_type">
        <Table name="pollutant"/>
        <Table name="typePollutant"/>
     </Join>
    </Hierarchy>
   </Dimension>
</Cube>
</Schema>
```

SCHEMA XML: DEFINIZIONE GERARCHIE - SNOWFLAKE

```
<?xml version="1.0"?>
<Schema>

 <Dimension name="Location" foreignKey="id_city">
   <Hierarchy primaryKey="pk_city" primaryKeyTable=
"location">
    <Join leftKey="fk_department" rightKey="pk_department">
      <Table name="Location"/>
        <Join leftKey="fk_region" rightKey="id_region">
          <Table name="Department"/>
          <Table name="Region"/>
        </Join>
      </Join>
  </Hierarchy>
 …
```

Level

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

```
<!ELEMENT Level (…)
<!ATTLIST  Level
 approxRowCount CDATA #IMPLIED
 name CDATA #IMPLIED
 table CDATA #IMPLIED
 column CDATA #IMPLIED
 nameColumn CDATA #IMPLIED
 uniqueMembers(true|false)>
```

SCHEMA XML: LIVELLI

```
<?xml version="1.0"?>
<Schema>
 <Dimension name="Location" foreignKey="id_city">
   <Hierarchy primaryKey="pk_city" primaryKeyTable=
"location">
    ……
   <Level name="Region" column="name_region"
    table="Region" uniqueMembers="true"/>
   <Level name="Department" column="name_department"
    table="Department"  uniqueMembers="true"/>
   <Level name="City" column="namecity"
      table="Location" uniqueMembers="true"/>
  </Hierarchy>
 …
```

Measure

- Name: nome misura
- Column: corrispondente colonna nella fact table
- 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.

```
<!ELEMENT Measure
  (Annotations?,MeasureExpression?,(CalculatedMemberP
  roperty)*)>
<!ATTLIST Measure
name CDATA #REQUIRED
column CDATA #IMPLIED
datatype
  (String|Numeric|Integer|Boolean|Date|Time|Timestamp
  ) #IMPLIED
aggregator CDATA #REQUIRED
visible (true|false) #IMPLIED>
```

SCHEMA XML: MISURE

```
<?xml version="1.0"?>
<Schema>
  <Cube name="Pollution">
  <Table name="pollution"/>
   <Dimension name="Pollutants" foreignKey="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"/>
                                …
   <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]

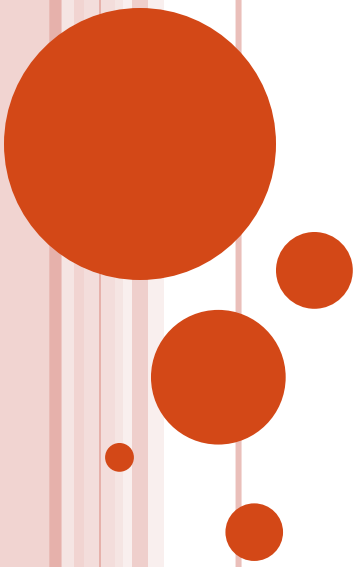
Esempio MDX

- Selezioniamo il nostro cubo Pollution attraverso la query MDX:

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

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

```
SELECT
   {[Measures].[PollutionValue]} ON COLUMNS,
   {[Location],[Polluttants]} 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:MONDRIANQUERY

- **<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=""
  jdbsPassword=""
  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= “SessioonObjectID”
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

- **<wcf: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=""/>
```

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

```
<wfc:render ref= "#{objectToRender}"
  xlsUri="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:modrianQuery` 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

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

<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/