INTERNAL REPORT

OACagliari VAMDC QChlTool Software and OACagliari VAMDC Node

Andrea Saba, Giacomo Mulas, Giuliano Malloci

Report N. 49, released: 28/05/2015

Reviewers: Sergio Poppi, Antonietta Angela Rita Fara

OAC Osservatorio Astronomico di Cagliari

OACagliari VAMDC QChITool Software and OACagliari VAMDC Node

Andrea Saba, Giacomo Mulas, Giuliano Malloci

28 maggio 2015

Indice

Sommario

Il gruppo di ricerca di Astrochimica dell'Osservatorio Astronomico di Cagliari si occupa, tra le altre cose, della modellizzazione teorica di grandi molecole organiche che si pensa possano essere presenti nello spazio, tra cui quelle della classe degli idrocarburi policiclici aromatici (PAH).

In questo documento verranno illustrati la progettazione e la struttura del database `pah_data` che ospita i dati prodotti dal gruppo di ricerca, il software QChITool (Quantum Chemical Import Tool) sviluppato nell'ambito del progetto VAMDC per la conversione e l'inserimento dei dati ottenuti come risultato di calcoli eseguiti con quantum chemistry software nel database `pah_data`, e l'adattamento a questo database della web application sviluppata dal consorzio dei partecipanti al progetto europeo VAMDC, chiamata nodo.

Verrà inoltre illustrato l'uso dell'interfaccia web per la navigazione dei dai dati presenti nel database. In particolare il software QChITool è stato progettato e sviluppato in riferimento ad alcuni specifici quantum chemistry software quali NWChem[2], Octopus[3] e Gaussian[4], pur mantenendo comunque un'impostazione generale che ne consenta un'estensione relativamente semplice per supportare, in futuro, altri pacchetti.

1 Premessa

Il progetto VAMDC Virtual Atomic and Molecular Data Center[5] è un progetto che è stato finanziato dall'Unione Europea nell'ambito del framework FP7 "Research Infrastructures - INFRA-2008-1.2.2 - Scientific Data Infrastructures".

Il progetto è partito il giorno 01/07/2009 ed ha avuto una durata di 42 mesi.

L'obbiettivo del progetto è stato quello di costruire un'infrastruttura digitale che facilitasse la localizzazione, l'accesso, la cross-correlation di dati atomici e molecolari accessibili su una molteplicità di banche dati eterogenee, distribuite in numerosi paesi.

Il progetto, che ha coinvolto 15 partner amministrativi composti da 24 gruppi provenienti da 6 stati europei, più la Serbia, la Federazione Russa e il Venezuela, ha messo insieme, da una parte ricercatori che si occupano di spettroscopia in diversi settori della fisica (e.g. astrofisica, fisica atmosferica, plasma), e dall'altra ricercatori nell'ICT con competenze delle e-infrastructure.

Il progetto era diviso in 8 Work Packages. Nei primi 3 di questi si sono svolte le attività di coordinamento dell'infrastruttura e sono stati coinvolti diversi progetti internazionali che potessero essere interessati al progetto VAMDC anticipando le sue potenzialità, nell'ottica di espandere al massimo le casistiche sia dei dati potenzialmente accessibili tramite VAMDC, sia dei suoi potenziali utilizzatori. Inoltre in questi WPs si sono prese tutte le decisioni di politica del progetto e di definizione delle policies dell'infrastruttura.

I WPs 4 e 5 sono stati dedicati alla progettazione e allo sviluppo dell'infrastruttura nelle sue componenti: dictionary, registry, node software, formato di interscambio dei dati (XAMS-VAMDC). Nei WP 6, 7 e 8 sono stati sviluppati un insieme completo di tools necessari per la creazione della piattaforma definendo delle nuove specifiche e creando, adattando e integrando dei nuovi software.

Nell'ambito del progetto, l'INAF si è impegnata con le sedi di Cagliari e di Catania. In particolare la sede di Cagliari ha avuto come obbiettivo l'adattamento dei suoi dati spettroscopici teorici su idrocarburi policiclici aromatici in

un formato adatto ad essere fruibile da un nodo del progetto opportunamente configurato.

In questo documento verrà illustrato come è stato progettato e implementato il database per ospitare i dati raccolti dal gruppo di ricerca, la progettazione o sviluppo del software QChITool per l'estrazione dei dati direttamente dagli output dei quantum chemistry software e il loro inserimento nel database e infine dell'adattamento del software del nodo sviluppato nell'ambito del progetto, per rispondere alle esigenze specifiche del team locale.

1.1 I quantum chemistry software

I quantum chemistry software sono pacchetti di chimica computazionale che su basi teoriche, calcolano da principi primi o tramite metodi semiempirici approssimati le proprietà e il comportamento di sistemi chimici (e.g. molecole e solidi), ottenendo proprietà fisiche di base (e.g. energie di legame, configurazioni di equilibrio etc.), e spettroscopiche (e.g. costanti rotazionali, modi propri di vibrazione, intensità delle transizioni vibrazionali e rotazionali, stati elettronici eccitati, spettro di fotoassorbimento elettronico verticale, etc.).

Esistono due principali gruppi relativi ai metodi utilizzati in chimica computazione e sono i metodi (semi)classici, che usano la meccanica classica per descrivere il moto degli ioni, in un campo di forze efficaci, e i metodi esplicitamente quantomeccanici. Questi ultimi possono essere a loro volta divisi in metodi *ab initio* e metodi semiempirici. I metodi *ab initio* risolvono esplicitamente l'equazione di Schrodinger[1], mentre quelli semiempirici semplificano drasticamente il calcolo evitando di risolvere esplicitamente parte dell'equazione di Schrödinger, assumendo di poterne approssimare le soluzioni con famiglie di funzioni parametriche relativamente semplici da trattare, i cui parametri sono ottenuti da fit a risultati sperimentali.

Tra i metodi *ab initio* l'approssimazione più semplice è quella di Hartree–Fock, che cerca la migliore approssimazione alla soluzione dell'equazione di Schrödinger elettronica sotto forma di un singolo determinante di Slater[2]. Approssimazioni un po' più accurate di quella di Hartree–Fock sono basati su tecniche perturbative, come il Møller–Plesset. Risultati decisamente più precisi sono possibili rappresentando la funzione d'onda elettronica sotto forma di combinazione lineare di più determinanti di Slater, usando tecniche diverse per troncare il numero di determinanti di Slater da includere nello sviluppo in serie, ottenendo così tutta una zoologia di metodi di livello migliore (e.g. MCSCF[6], Coupled Cluster[7] e così via). Questi metodi, almeno in linea di principio, convergono alla soluzione esatta al crescere della base usata, ma hanno uno scaling molto ripido con il numero di particelle del sistema, diventando rapidamente ingestibili per sistemi con più di qualche decina di atomi pesanti (non idrogeno). Un metodo in principio esatto, ma computazionalmente molto pesante è il Monte Carlo quantistico[9], al momento applicabile solo a sistemi di pochi elettroni ma promettente in prospettiva grazie allo scaling del costo computazionale con le dimensioni del sistema molto meno ripido dei metodi multireference. Una men-

[1] In meccanica quantistica non relativistica l'equazione di Schrodinger è un'equazione fondamentale che determina l'evoluzione temporale dello stato di un sistema, ad esempio di una particella, di un atomo o di una molecola

[2] un prodotto di funzioni di una sola particella antisimmetrizzata per scambio di particelle identiche

zione a parte merita la Teoria del funzionale densità (DFT)[8], che si distingue per il fatto di non basarsi sull'equazione di Schrödinger; si basa infatti su una serie di teoremi matematici che dimostrano sostanzialmente che lo stato di un sistema quantistico di particelle interagenti può essere completamente determinato dalla loro sola densità (senza conoscere la funzione d'onda a molti corpi), e che si può sempre trovare un sistema ausiliario fittizio di particelle non interagenti, immerse in un potenziale efficace, che hanno la stessa densità del sistema interagente. In questo modo , noto il potenziale efficace, il sistema ausiliario si può risolvere con equazioni formalmente simili a quelle di Hartree–Fock, dette di Köhn e Sham, molto meno "costose" computazionalmente dei metodi più accurati, ottenendo comunque una soluzione in principio esatta. Il problema, in questo caso, è la determinazione del potenziale efficace, detto funzionale di scambio–correlazione, di cui non è nota la forma esatta e che viene rappresentato con varie approssimazioni.

1.2 Gli idrocarburi policiclici aromatici (PAH)

Gli idrocarburi policiclici aromatici sono dei composti organici che contengono soltanto atomi di carbonio e di idrogeno che presentano due o più anelli aromatici[3].

I PAH sono studiati in astrochimica perché sono ritenuti una componente importante delle nubi di gas molecolare. Intense emissioni infrarosse alle frequenze tipiche dei modi di vibrazione di questa classe di molecole sono osservate dovunque materiale interstellare sia illuminato da radiazione ultravioletta, e la curva di estinzione interstellare presenta un andamento, nell'ultravioletto, molto simile al loro spettro medio di assorbimento. Lo studio sperimentale delle proprietà spettroscopiche di queste molecole in fase gassosa è piuttosto difficile, data la loro refrattarietà e la loro reattività nelle condizioni di ionizzazione previste per loro nello spazio. L'utilizzo di software di chimica quantistica permette di studiare in maniera sistematica, su larga scala, un campione di specie chimiche molto più grande di quello che sarebbe praticabile per via sperimentale. Può dunque essere usato sia per guidare la selezione di un numero ristretto di specie su cui vale la pena di investire lo sforzo per uno studio sperimentale e, una volta ottenuto quest'ultimo, per l'identificazione delle features spettroscopiche osservate.

1.3 Struttura del report

Il presente report presenterà prima di tutti i passi che hanno portato alla progettazione del database che conterrà i dati prodotti dal gruppo di ricerca. Questo database è inteso come risorsa a se stante indipendentemente dallo sviluppo del nodo del progetto VAMDC, del QChITool e del web front end.

Viene quindi presentato la web application QChITool per l'inserimento dei dati nel database. Questa è stata sviluppata per essere fruibile come web application ma anche come applicazione indipendente anche da riga di comando.

Di seguito viene presentata la sezione relativa all'architettura del progetto VAMDC tramite i suoi standard e i suoi protocolli. Questa infrastruttura

[3]Un anello aromatico è un sistema ciclico a struttura planare in cui tutti gli atomi coinvolti condividono tramite i loro orbitali p un totale di $4n + 2$ elettroni, dove n è un intero positivo

è studiata per essere indipendente dalle singole implementazioni dei nodi del consorzio.

Nella sezione successiva vengono illustrate le modifiche che non state apportate al nodo VAMDC standard messo a disposizione per il progetto, per renderlo conforme alle esigenze del gruppo di astrochimica dell'Osservatorio Astronomico di Cagliari. Il Nodo si interfaccia al database pah_data tramite una precisa definizione del suo schema.

Nell'ultima sezione è presentato il frontend web del database che è indipendente sia dal QChITool che dal nodo VAMDC e dall'architettura del progetto. Questo è stato sviluppato cogliendo l'occasione del progetto VAMDC per prendere il posto del precedente sito web che metteva a disposizione i dati del gruppo di astrochimica dell'Osservatorio Astronomico di Cagliari in modalità statica e non basato su database.

2 Progettazione del database pah_data

Per la progettazione del database relazionale si è partiti dal documento realizzato dal gruppo di ricerca e allegato al presente documento. Come motore di database è stato scelto MySQL 5.5 [10] per la licenza di utilizzo, le possibilità di scalabilità e quelle di mirroring. Lo schema del database in dettaglio è presente in allegato.

Le entità principali dello schema relazionale sono: task, electronicstates, electronictransitions, vibrationanalisesarmonic, vibrationanaliseshanarmonic.

In Fig. 1 lo schema semplificato.

Il task identifica la singola operazione eseguita in un calcolo. Questo è corredato di tutte le informazioni che sono state impostate per il calcolo come per esempio il basis set[4].

L'electronicstate definisce lo stato elettronico di una determinata molecole identificata dalla sua formula chimica e dalla posizione spaziale di ogni singolo atomo che la compone.

Le electronictransitions sono una coppia di stati elettronici diversi di una stessa molecola.

Agli electronicstates vengono inoltre associate altre informazioni, se calcolate, quali il momento di dipolo elettrico, la polarizzabilità, l'energia di ionizzazione, le costanti rotazionali.

Per quanto riguarda i calcoli è stato scelto di salvare direttamente i file elencati in precedenza in un record del database in formato compresso. Questo permette di archiviare tutto il contenuto del database con un unico dump che può essere inviato alla sede dell'IRAP di Tarbes e all'Observatoire de Paris–Meudon, in cui sono presenti repliche del database e del nodo VAMDC.

3 Quantum Chemistry Import Tool (QChITool)

Il Quantum Chemistry Import Tool (QChITool) è una web application realizzata tramite il framework Django 1.4.5[1] che permette il parsing dei files di output di

[4]L'insieme di funzioni di base che sono state utilizzate come base per l'espansione della funzione d'onda considerata

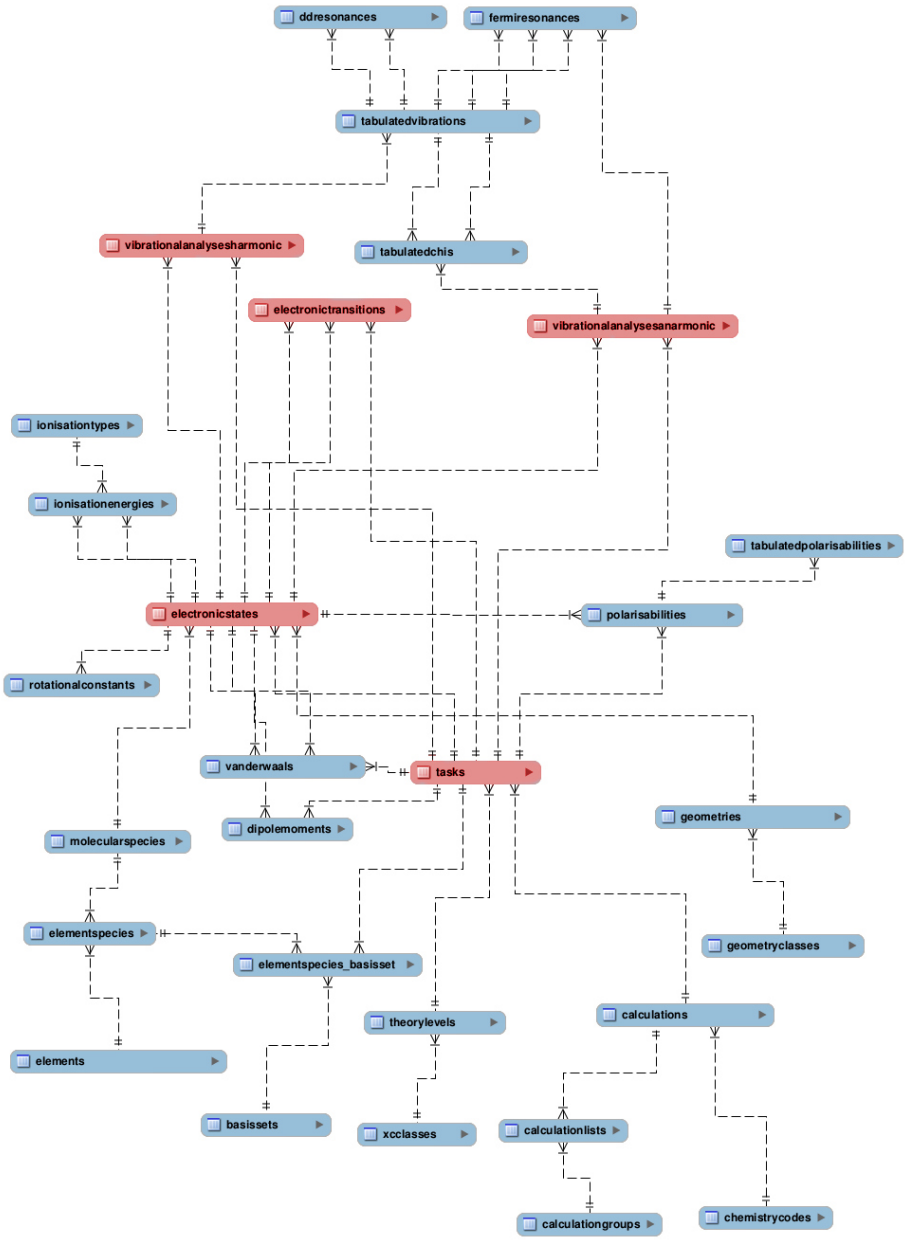

Figura 1: Database pah_data schema

alcuni quantum chemistry software come NWChem, Octopus, Gaussian, etc. Il risultato del parsing può essere salvato in un database relazionale. Nella versione attuale (0.7) l'unico quantum chemistry software supportato è NWChem sino alla versione 6.1.1

L'applicazione permette l'inserimento di nuovi dati nel database, l'amministrazione dei dati presenti nel database tramite l'interfaccia amministrativa, l'utilizzo di alcune funzioni globali sul database che permettono un riallineamento delle informazioni presenti.

Ogni connessione alla web application viene identificata da Django tramite una chiave di sessione univoca. Quando si procede all'inserimento di nuovi dati il QChITool esegue tre fasi.

Il diagramma di flusso dettagliato dell'applicazione è riportato in Fig. 2.

La prima fase consiste nell'upload del file di input, del file di output e del file di database risultanti da un calcolo eseguito con NWChem. I files vengono salvati in una cartella che ha per nome l'MD5 della chiave di sessione, questa cartella viene posizionata nella cartella definita in `settings.MEDIA_ROOT`. A questo punto viene chiamata la funzione `[1].return_parsed_object_for_session` [5] alla quale vengono passati i path dei file che sono stati salvati nella fase precedente.

La funzione `return_parsed_object_for_session` chiama la `[2].parsing` che si occupa del parsing vero e proprio del file e lo utilizza, tramite la funzione `[1].build_calculation_object`, per creare gli oggetti corrispondenti a quelli che saranno i record del database.

La funzione `parsing` divide il file di output in righe tramite l'oggetto `MappingSections`.

Prima di procedere all'operazione di parsing vera e propria viene eseguita la funzione `[2].clean_output_file_rows` che si occupa della pulizia del file di output eliminando dal file alcune righe che vengono aggiunte come messaggi del calcolo per l'utente, in posizioni imprevedibili a priori. Le regular expressions che definiscono l'inizio e la fine di queste porzioni di file si trovano nella lista `[3].cutter_extremes` come liste in cui il primo elemento è la regular expression iniziale e il secondo quella finale. La definizione di queste stringhe deve essere la meno ambigua possibile e deve corrispondere solo con la porzione di testo cercata.

La funzione inoltre sostituisce la coppia di caratteri "`\r\n\`", generata dall'importazione del file da un campo del database, con il carattere "`\n`". Per concludere, la funzione divide il file in una lista di righe separate dal carattere "`\n`".

Per delimitare l'inizio e la fine di ogni sezione viene fatto il confronto di ogni riga con tutte le regular expression dei `MappingObjects` di primo livello, se viene trovata una corrispondenza viene creato un oggetto `[4].SectionFound` ed aggiunto alla lista delle sezioni. Vengono quindi cercate le corrispondenze con i `MappingObjects` del livello successivo e in caso positivo viene creato un nuovo `SectionFound` object. Se viene trovata una corrispondenza vengono cercate le corrispondenze con i `MappingObjects` del terzo ed ultimo livello. Al temine di ogni ricerca viene utilizzata la riga precedente all'ultima corrispondenza per chiudere la precedente `SectionFound` trovata.

[5] In appendice la codifica dei prefissi

3.1 La classe `MappingSections`

Nella classe `[7].MappingSections` sono elencati come lista di `MappingObjects` le sezioni conosciute dei files di output e del database.

I `MappingObjects` sono definiti come:

```
class MappingObject:
    def __init__(self, name, re_ex, relatedclass):
        self.name = name #description name of the section
        self.re_ex = re_ex #regular expression for the first line
     of the section
        self.relatedclass = relatedclass #Name of the class that
    manage this section
        self.re_ex_compiled = re.compile(self.re_ex) #compiled
    regular expression for faster parsing
```

L'idea è quella di confrontare tutte le righe del file da analizzare con la regular expression `re_ex` per identificare l'inizio di una sezione. Per velocizzare l'esecuzione la regular espression viene salvata anche nella sua forma compilata nell'attributo `re_ex_compiled`. Infine, tramite `relatedclass`, viene associata al `MappingObject` la classe che si occuperà di estrarre dalla sezione i dati trovati.

Le sezioni sono diverse per i due tipi di file sul quale può essere effettuato il parsing (nel caso di NWChem il file di output e il file di database).

La classe `MappingSections` crea una lista dei `MappingObjects` a seconda del parametro passato al costruttore che può essere `NwChemOutput` o `NwChemDatabase`.

```
class MappingSections:
        def __init__(self, filetype):
            self.levels = []
            if filetype == "NwChemOutput":
                self.levels.append([MappingObject("arguments", r"
    argument  1 =.*", nwc_level_0_sections.Arguments),
                                    MappingObject("echo", r"[=]+ echo
    of input deck [=]+",  nwc_level_0_sections.InputDeck),
                                    MappingObject(...),
                                              MappingObject(...),
                                    ])
                self.levels.append([MappingObject("Memory
    information", r"Memory information",  nwc_level_2_sections.
    MemoryInformation),
                                    MappingObject(...),
                                             ...
                                    MappingObject(...),
                                    MappingObject(...),
                                    ])
            elif filetype == "NwChemDatabase":
                self.levels.append([MappingObject("arguments", r"
    argument  1 =.*", nwc_level_0_sections.Arguments),
                                    MappingObject(...),
                                    MappingObject(...),
                                    ...
                                    MappingObject(...),
```

```
])
```

Nel caso del `NwChemOutput` i `MappingObjects` sono raggruppati in una lista di liste che rappresenta l'organizzazione gerarchica delle sezioni presenti nel file di output. Ogni sezione all'interno del file può essere ripetuta e all'interno di ognuna di queste sezioni si può trovare una ripetizione delle sezioni del livello sottostante e così via. In ogni lista l'ordine dei `MappingObjects` è quello in cui verranno confrontate le righe iniziali delle sezioni nel file di output e nel caso in cui una riga corrisponda alla regular expression relativa ad un `MappingObject`, quelli successivi verranno ignorati.

3.2 La classe `SectionFound`

Lo schema della classe `SectionFound` è

```
class SectionFound:

    def __init__(self, section_name, striped_file_row,
    section_initial_line, section_final_line,
    section_final_information_line, section_class):
        self.section_name = section_name
        self.striped_file_row = striped_file_row
        self.section_initial_line = section_initial_line
        self.section_final_line = section_final_line
        self.section_final_information_line =
    section_final_information_line
        self.section_class = section_class
        self.section_object = None
        self.list_of_subsections = []

    def print_me(self):
        print "section_name = " + self.section_name
        print "striped_file_row = " + self.striped_file_row
        print "section_initial_line = " + str(self.
    section_initial_line)
        print "section_final_information_line = " + str(self.
    section_final_information_line)
        print "section_final_line = " + str(self.
    section_final_line)
        print "section_class = " + str(self.section_class)
        for sec in self.list_of_subsections:
            sec.print_me()

    def __unicode__(self):
        return "SectionFound Object:" + "section_name = " + self.
    section_name
```

dove il metodo `print_me()` e il metodo `__unicode__()` hanno solo funzione di debugging.

Gli attributi della classe sono:

```
self.section_name = section_name
```

è il nome della sezione presa dal `MappingObject` corrispondente

```
self.striped_file_row = striped_file_row
```

è la riga iniziale della sezione che è anche quella per la quale è stata trovata la corrispondenza con la regular expression del `MappingObject`

```
self.section_initial_line = section_initial_line
```

L'indice della prima riga della sezione rispetto al file di output

```
self.section_final_line = section_final_line
```

L'indice dell'ultima riga della sezione

```
self.section_final_information_line =
    section_final_information_line
```

L'indice dell'ultima riga della sezione in cui ci sono delle informazioni "utili"

```
self.section_class = section_class
```

il nome della classe che gestirà l'estrazione dei dati dalla sezione. Preso dal `MappingObject`

```
self.section_object = None
```

l'oggetto creato tramite `self.section_class`

```
self.list_of_subsections = []
```

la lista degli oggetti `SectionFound` corrispondenti alle sotto sezione dell'attuale sezione.

Dopo aver creato tutti gli oggetti `SectionFound` la funzione `parsing` esegue la funzione `set_final_information_line()` che assicura la chiusura di tutte le sezioni con il settaggio della loro riga finale. Infine la funzione `parsing` si occupa di create per ogni oggetto `SectionFound` l'attributo `section_object` tramite la `section_class` che è stata recuperata dal `MappingObject`. Al costruttore verranno passati gli indici iniziali e finali della sezione compresi quelli che indicano l'inizio e la fine della informazioni "utili" e la lista delle righe che costituiscono la sezione.

La funzione restituisce la lista di tutte le `SectionFound` che per costruzione sono organizzare gerarchicamente tramite l'attributo `list_of_subsections` di ogni `SectionFound`.

Organizzazione delle classi che definiscono le sezioni conosciute. (`nwc_sections`)

Nei files `[8].nwc_level_0_sections`, `[8].nwc_level_1_sections` e `[8].nwc_level_2_sections` sono definite le classi che si occupano di estrarre i dati dalle sezioni individuate tramite i `MappingObjects`.

Queste classi sono tutte ereditate dalla classe `[4].GenericSection` dove oltre il costruttore e la funzione per il debugging `print_section` sono definite le funzioni:

```
go_to_first_line(self, file_pointer = None)
```

che restituisce l'indice della prima riga della sezione e la prima riga della sezione

`split_section(self, list_of_subsection_re)`

che divide la sezione in sottosezioni, utilizzando come regular expression per identificare l'inizio di ogni sottosezione, l'argomento `list_of_subsection_re`.

La funzione restituisce una lista di `SectionFound`.

`return_row(self, line_number)`

che restituisce la riga che ha indice `line_number`. L'indice è relativo al file di output.

`find_value(self, re_valuename, separator, re_compiled, search_in_all_section)`

restituisce la prima occorrenza della variabile identificabile dalla regular expression compilata `re_compiled`. La variabile booleana `search_in_all_section` definisce se la corrispondenza verrà cercata in tutta la sezione o solo nella parte della sezione delimitata dalla variabile `section_final_information_line`. L'argomento `separator` è la stringa che separa il nome della variabile dal suo valore. La funzione restituisce la lista `[nome variabile, valore]`.

Le `nwc_sections` definiscono un costruttore che invoca quello della classe padre e che imposta la variabile `values` dell'oggetto. Questa variabile è sempre un dictionary dei valori trovati.

Le classi presenti in `nwc_sections` effettuano il parsing di sezioni nelle quali si presentano molto spesso degli schemi di dati ricorrenti. Le funzioni corrispondenti al parsing dei diversi tipi di schemi di dati, sono state raccolte nella libreria `qchitool.parser.tools.parser_tools`

Queste funzioni sono:

`return_file_rows(filename, initial_line, final_line)`

che restituisce una lista delle righe comprese tra `initial_line` e `final_line` del file `filename`. Nel caso in cui `final_line` sia nullo la funzione restituisce un oggetto lista con la sola `initial_line`.

`openbabel_count_hetero_rings(mol)`

che restituisce il numero di cicli aromatici della molecola passata. La molecola è di tipo `OBMol` e la funzione utilizza il metodo `GetSSSR()` `http://openbabel.org/dev-api/classOpenBabel_1_1OBMol.shtml`

`def chunks(l, n):`

che divide la generica lista `l` in liste da `n` elementi ognuna. L'ultima lista può avere un numero di elementi minore o uguale a `n`.

Example

```
[input] [2, 3, "d", 0.8, 33, 1, 99], 2
[output] [[2, 3], ["d", 0.8], [33, 1], [99]]
```

`separate_by_schema(splited_line, schema, description_field, value )`

che prende come input una lista di stringhe `splited_line` che è una riga già divisa in tokens e per ogni token associa il tipo di dato definito dallo schema della stringa `schema`. L'argomento `description_field` l'indice in base `0` del primo campo con la descrizione della riga. Nel caso in cui la lunghezza della stringa `schema` sia inferire al numero di elementi della lista `slited_line` la descrizione viene estesa per un numero di campi uguale alla differenza tra le due lunghezze. La nuova riga divisa in elementi viene aggiunta alla lista `value` come tupla. `schema` è composto da un numero arbitrario di elementi dell'alfabeto `{"s", "n"}` dove "`s`" indica una stringa, e "`n`" indica un valore numerico.

Example

```
[input]  - ['2', 'C', '6.0000', '-0.57776201', '-0.43712846',
   '0.00000000'], 'nsnnnn', 1, [...]
[output] - [..., ([2, 'C', 6.0, -0.57776201000000005,
   -0.43712846, 0.0])]
```

`get_parenteses_value(row, parenteses)`

che restituisce la stringa compresa tra le prime occorrenze delle due stringhe presenti nella lista `parenteses`.

`get_singlevalue_sep(row, tuple_separator, value_separator, only_one_value, mydictionary)`

che divide la riga `row` tramite la stringa `tuple_separator`. Ognuno di questi token viene diviso tramite la stringa `value_separator`. Le coppie ottenute vengono aggiunte al dictionary `mydictionary`. Nel caso in cui l'argomento booleano `only_one_value` sia impostato come `True` e non sia stato impostato l'argomento `tuple_separator`, viene salvata solo la prima coppia trovata.

Example

```
[input] - "Use of symmetry is: on ; symmetry adaption is: on ",
    ";", ":", False, {...}
[output] - mydictionary = {..., "Use of symmetry is": "on", "
    symmetry adaption is": "on"}
```

`get_listvalue_sep(filename, initial_line, end_line, tuple_separator, value_separator, only_one_value, section_rows = None)`

che sfrutta la funzione `get_singlevalue_sep` per estrarre i valori da una lista di righe.

`get_listvalue_no_sep(filename, initial_line, end_line, section_rows = None)`

funzione che prende come argomento il nome del file e gli indici iniziale e finale delle righe del file dal quale estrarre i dati. Eventualmente è possibile passare direttamente la lista delle righe con l'argomento `section_rows`. La funzione restituisce un dictionary con la lista dei parametri nella forma `"descrizione(with spaces) valore numerico"` eventualmente seguiti dalla stringa `"local"` or `"non-local"`.

Example

```
[input] - ["No. of variables    17199", "Convergence  1.0E-04
    local"]
[output] - {"No. of variables": 17199, "Convergence(local):  1.0E
    -04}
```

```
get_value_from_to_no_sep(filename, initial_line, end_line,
    separator, section_rows = None)
```

funzione che restituisce i valori estratti dalla riga nella forma `"descrizione(with spaces) from valorenumerico1 to valorenumerico2"` dove la stringa `"to"` è l'argomento `separator` passato alla funzione. Example

```
[input] - ' Renormalizing density from     120.00 to    119'
[output] - [['Renormalizing density from', 120.0, 119]]
```

```
get_multirow_table(filename, initial_line, end_line, schema,
    skip_first_row, section_rows = None)
```

Funzione che estrae i dati da una lista di righe passata (`section_rows`) o da file `filename` iniziando dalla riga di indice `initial_line` e finendo alla riga di indice `end_line`. Ogni riga segue lo schema `schema` eventualmente saltando la prima riga se è impostato a `True` l'argomento `skip_first_row`. La funzione restituisce una lista di tuple una per ogni riga.

```
def get_continous_list_no_sep(filename, initial_line, end_line,
    schema, section_rows = None)
```

Funzione che estrae i dati da una lista di righe passata (`section_rows`) o da file `filename` iniziando dalla riga di indice `initial_line` e finendo alla riga di indice `end_line`. Le righe vengono concatenate per formare un unica riga e i dati vengono estratti secondo lo schema `schema`. Se lo schema ha una lunghezza maggiore di `1` la funzione restituisce una lista di tuple ognuna secondo lo schema, altrimenti la funzione restituisce la lista dei valori trovati del tipo corrispondente all'unico carattere della stringa schema.

```
get_tabulated_data(filename, initial_line, end_line, schema,
    first_row_value, description_field, value_separator,
    section_rows = None)
```

Funzione che estrae i dati da una lista di righe passata (`section_rows`) o da file `filename` iniziando dalla riga di indice `initial_line` e finendo alla riga di indice `end_line`. La prima riga dalla quale estrarre i dati deve iniziare con il valore `first_row_value`. La stringa `value_separator` può essere `None` o vuota. I dati dalle singole righe verranno estratti dalla funzione `separate_by_schema` alla quale verranno passati `schema` e `description_field` secondo le specifiche della funzione `separate_by_schema`. La funzione restituisce una lista di tuple secondo lo schema `schema`.

```
get_z_matrix(filename, initial_line, end_line, section_rows =
    None)
```

Funzione che costruisce dalla tabella della Z matrix una lista di tuple secondo schemi diversi a seconda che la riga contenga informazioni di "Stretch", "Bend" o "Torsion".

```
get_hessian_derivative_value(filename, initial_line, end_line,
    section_rows = None)
```

Funzione che estrae i dati dalle componenti delle derivate parziali del momento di dipolo. Restituisce una lista di oggetti `[5].EnergyForDerivativeDipole`. Questi oggetti memorizzano il vettore, l'atomo, la direzione, la carica e l'unità (es. debye/angstrom).

Example:

```
Y vector of derivative dipole (au) [debye/angstrom]
 d_dipole_y/<atom=   1,x> =    -0.5165     [   -2.4810]
 d_dipole_y/<atom=   1,y> =    -0.3177     [   -1.5258]
 d_dipole_y/<atom=   1,z> =     0.0000     [    0.0000]
 d_dipole_y/<atom=   2,x> =     0.5165     [    2.4810]
```

```
get_triangular_matrix(filename, initial_line, end_line,
    section_rows = None)
```

Funzione che legge i dati di una matrice triangolare e restituisce un dictionary che per l'indice delle righe memorizza una lista dei valori della riga.

```
return_inputoutput_sectionschema(parsing_result)
```

Lo schema dei dati che si trovano in un file di output di NWChem può essere considerato come la risposta alla esecuzione di uno o più tasks. Per ognuno di questi task è presente un modulo di input e tutti i moduli relativi all'output del calcolo. Questa funzione restituisce una lista degli oggetti `[5].InputOuputSection` uno per ogni task.

```
returnsection(section_name, parsing_result, number)
```

Funzione che cerca una sezione con un determinato nome in una lista di `[4].SectionFound` objects. L'output della funzione sarà differente a seconda del valore passato come argomento `number`:

ALL La funzione restituirà una lista di tre liste. Le liste corrispondono ai `MappingObject` di primo, secondo e terzo livello. In ogni lista saranno presenti tutti i `SectionFound` corrispondenti alla stringa `section_name` passata come argomento.

LIST La funzione restituisce una lista di tutti i `SectionFound` corrispondenti alla stringa `section_name` passata come argomento.

FIRST and LAST La funzione restituisce un solo `SectionFound` corrispondente, rispettivamente, alla prima o all'ultima occorrenza delle `SectionFound` corrispondenti alla stringa `section_name` passata come argomento.

```
writefile(lines, restart_name, session_path, format)
```

Crea un nuovo file con il nome `"_" + format + "_" + restart_name + format` nella directory `session_path`. Nel caso in cui questo argomento sia nullo allora salva il file nella directory temporanea di default. Nel file vengono scritte le righe `lines` passate e il file viene chiuso. La funzione restituisce il nome completo del file.

```
extract_molecule_info(out_file, charge, session_path,
    basis_set_list, geometry_list, spin_multiplicity,
    unit_conversion_value)
```

La funzione costruisce una molecola come nuovo oggetto `pybel.ob.OBMol()` con gli argomenti passati. La funzione inoltre crea tre file con il nome `out_file` nella directory `session_path`. Se l'argomento `session_path` è `None` i files vengono cercati nella directory temporanea di default. I tre files avranno estensione "xyz", "cml" e "sdf" e in questi files sarà presente la geometria della molecola creata nei tre formati corrispondenti alle estensioni.

`basis_set_list` è una lista di coppie `["nome elemento", "basis sets name"]` che indica per ogni elemento quale basis set è stato utilizzato. Nel caso in cui `nome elemento` sia uguale a "`*`" allora il `basis set name` è stato utilizzato per tutti gli elementi della molecola. `Charge` indica la carica totale della molecola.

In `geometry_list` è presente la lista degli atomi con le relative coordinate cartesiane, il numero atomico e il simbolo. Per convertire le coordinate nell'unità di misura voluta queste vengono divise per il fattore di conversione `unit_conversion_value`.

Nella nuova molecola viene utilizzato il metodo `NewAtom()` per aggiungere un atomo, il metodo `SetVector()` per posizionarlo, il metodo `SetAtomicNum()` per impostare il numero atomico di ogni atomo. Dopo che sono stati creati tutti gli atomi della molecola, con il metodo `ConnectTheDots()` vengono connessi tramite singoli legami in base alla prossimità e alla valenza degli atomi.

Vengono quindi creati i file con le geometrie nei formati "xyz" e "cml". Con il metodo `PerceiveBondOrders()` vengono invece creati i doppi legami della molecola e a questo punto viene creato il file con la geometria nel formato "sdf".

Per evitare che il nome della molecola venga impostato arbitrariamente da `OpenBabel`, dal file cml viene eliminata questa informazione.

Viene utilizzato `OpenBabel` per costruire l'inchi e l'inchikey a partire dal file di output di NWChem.

La funzione restituisce un oggetto di tipo `[5].MoleculeInfo` dove vengono memorizzate le informazioni ricavate.

Il risultato della funzione `parsing` viene passato come argomento `parsing_result` alla funzione `[1].build_calculation_object`, insieme al path dei files di input, output e database che sono stati salvati nella cartella temporanea del server.

Nella funzione `build_calculation_object` tramite la funzione `parser_tools.returnsection` vengono estratte le sezioni volute dal `parsing_result` e tramite la funzione `parser_tools.return_inputoutput_sectionschema` vengono estratte le sole sezioni di input-output dal `parsing_result`. Queste ultime vengono salvare in `input_output_sections`.

Dopo che sono state cercate le sezioni `NwchemManifest` e `JobInformation`, vengono letti i file di input e di output per poter essere salvati come BLOB di testo nel database.

La funzione `[1].create_human_readable_database_file` converte il file di database dal formato nativo di NWChem in un formato human readable. Questa funzione utilizza la versione di NWChem installata sul sistema operativo con

la direttiva `"task rtdbprint"`. Il file così creato verrà salvato dalla funzione nella cartella temporanea.

La funzione `build_calculation_object` restituisce un dictionary (`TableDataFound`) in cui saranno presenti zero o più degli oggetti relativi o alla struttura del database o al calcolo eseguito in base a quello che è stato trovato nel file di output. Le voci del dictionary sono: "ChemistryCodes", "Calculations", "Tasks", "TheoryLevels", "MolecularSpecies", "ElectronicStates", "DipoleMoments", "Elements", "RotationalConstants", "VibrationalAnalysesHarmonic", "TabulatedVibrations", "Geometries", "molecule_info", "optimization_convergence", "energy_analisys", "frequency_analisys", "property_analisys", "SwapOrbitals".

In generale per "ChemistryCodes" e "Calculations" avremo un solo oggetto ciascuno con le informazioni generali sul calcolo e i file di input output e database, per esempio versione, data, nome del file, md5 dei tre files.

Per i "Tasks" e i "TheoryLevels" avremo un oggetto per ogni task eseguito nel calcolo. A seconda del tipo del singolo Task verranno estratte e salvate le informazioni in "optimization_convergence", "energy_analisys", "frequency_analisys", "property_analisys". Per ognuno dei Task verrà aggiunto un elemento alle liste "MolecularSpecies", "ElectronicStates", "DipoleMoments", "Elements", "RotationalConstants", "VibrationalAnalysesHarmonic", "TabulatedVibrations", "Geometries", "SwapOrbitals" eventualmente nullo.

Nel caso di "Elements" and "TabulatedVibrations" averemo inoltre delle liste di liste.

L'unica nota è quella relativa alla lista "Geometries", le cui informazioni vengono estratte dal file output, ma nel caso in cui questo non sia possibile (tipicamente nel caso di restart del calcolo) i dati della geometria vengono estratti dal file di database convertito nella versione human readable. Per determinare se la geometria è già presente nel database (a meno di una rototraslazione) è necessario utilizzare la funzione `qchitool.tools.geometry_tools.similargeometry`, la cui descrizione è presente più in basso.

Con queste informazioni la funzione `[1].return_parsed_object_for_session` può svolgere due compiti importanti. Il primo è quello di controllare se nel database è già presente la molecola e/o lo specifico calcolo che si vuole inserire e in caso affermativo quali tasks del calcolo sono già presenti. La seconda costruire i tasks per poterli presentare all'utente e chiedergli quali vuole salvare nel database. Per svolgere la prima operazione viene chiamata `[6].count_calculation_by_md5` che cerca tutti i calcoli nel database che hanno lo stesso md5 per il file di output, e in caso affermativo restituisce anche tutti i task che sono presenti nel database per quel calcolo. Per la seconda operazione viene costruita una lista di oggetti `[5].TaskCode` che memorizza le informazioni del singolo task.

3.3 Gestione delle geometrie degli stati elettronici delle molecole

Ogni stato elettronico di una molecola è caratterizzato da una sua geometria che definisce la posizione spaziale degli atomi che la compongono prendendo in considerazione solo la massa atomica e il numero atomico dell'atomo. Può capitare quindi che lo stesso stato elettronico di una molecola abbia n descrizioni diverse in coordinate cartesiane della posizione degli atomi. Possiamo quindi dire che lo stato elettronico è lo stesso a meno di una roto-traslazione. Per

riportare due stati stati elettronici allo stesso sistema di riferimento procederemo quindi con il calcolo del centro di massa dei due stati elettronici e al calcolo degli autovettori che saranno la nuova base degli stati elettronici. Nel caso in cui due o tre dei tre autovettori siano uguali verranno prese in considerazione anche le riflessioni sugli assi corrispondenti.

Inoltre la posizione degli atomi può presentare un errore numerico o avere una precisione decimale che non ha alcun significato fisico. Per ovviare a questo problema è stato previsto che i due atomi corrispondenti di due stati elettronici vengano considerati nella stessa posizione a meno di un fattore d'errore. In questo modo le geometrie degli stati elettronici che vengono estratte dai file di output del programma di calcolo, sono raggruppate in classi di equivalenza.

L'applicazione è ora in grado di presentare all'utente le informazioni che è riuscita ad estrarre dai file che sono stati passati.

Nella pagina informativa è indicato se il calcolo è stato trovato nel database, e la lista dei task del calcolo. Per ogni task è indicato se la la molecola è già presente nel database, la geometria in formato CML, la Formula, l'Inchi, l'InchiKey, il numero di cicli aromatici, e la carica elettronica della molecola. Nel caso in cui la funzione sia stata abilitata sarà visualizzato anche il Jmolapplet 12 in cui sarà presente la molecola tridimensionale interattiva.

Ognuno dei task presenta un checkbox per poter indicare quali task si vuole che vengano salvati nel database.

La funzione `qchitool.views.nwc_calculus_information_to_save` si occupa di costruire un form con tutte le informazioni che potranno essere salvate nel database. La maggior parte dei campi dei singoli record viene completata con le informazioni estratte dal file di output. Sarà compito dell'utente completare le restanti informazioni come per esempio le fonti bibliografiche da associare ai records.

Svolti alcuni controlli di coerenza dei dati inseriti, questi vengono salvati nel database e nell'eventualità in cui siano presenti più stati elettronici nel database per la stessa molecola, viene chiesto all'utente di indicare quali sono quelli in cui la configurazione di energia è un minimo.

3.4 Funzioni aggiuntive per manutenzione del database

Le funzionalità aggiuntive che sono state introdotte sono:

Rebuild Geometries Classes Table Cancella le classi di geometria presenti nel database e le ricostruisce tutte.

Rebuild Functional Classes Table Cancella le classi Funzionali di scambio e correlazione presenti nel database e le ricostruisce tutte.

Delete all ionization energies Elimina le energie di ionizzazione degli stati elettronici

Recalculate ionization energy Ricalcola e crea le energie di ionizzazione degli stati elettronici

Rebuild molecular's elements and basis sets Cancella gli elementi presenti per ogni molecola e i relativi basis sets e li ricostruisce sulla base del file di output (o eventualmente sul file di database) presente nel database per il calcolo associato.

Alcune di queste funzionalità richiedono molto tempo di calcolo.

L'interfaccia amministrativa del database è quella messa a disposizione da Django con l'introduzione di alcune funzionalità specifiche per alcune tabelle. Una di queste è quella relativa alla cancellazione di dati dal database seguendo la convezione illustrata nel paragrafo cancellazione dei dati dal database

3.5 Cancellazione dei dati dal database

Le specifiche stabilite dal database prevedono che sia garantita la riproducibilità, ovvero che sia possibile chiedere al sistema quale sarebbe stata la risposta ad una query effettuata in un momento ben definito (per esempio "quale sarebbe stata la risposta se ti avessi fatto questa domanda il 15 aprile 2001 alle 17:00 ora centrale europea?"). Non deve essere quindi possibile cancellare i dati dal database senza che questi lascino una traccia della loro presenza, ma solo indicare che un certo dato da un certo momento in poi non deve essere più considerato valido, e quindi ignorato per tutte le queries fuori dal suo periodo di validità. Tutti i records sono quindi stati completati con un campo "expired data" di tipo `datetime` che indica il momento a partire dal quale quel record deve essere considerato non presente nel database. Questo campo insieme al campo "timestamp" che indica quando è stato inserito il dato nel database permettono di stabilire quali dati sono validi in tutta la storia dei database.

4 Architettura del progetto VAMDC

L'infrastruttura è composta principalmente da due web services: il registry e i nodi. A questi si aggiungono un portale e un servizio di monitoraggio. Il registry e i nodi hanno rispettivamente i compiti di coordinamento dei nodi e di risposta alle richieste di dati.

Il registry si occupa di mantenere una lista dei nodi che fanno parte dell'infrastruttura con una lista dei dati che ogni nodo è capace di mettere a disposizione (capabilities). Questa lista di capacità è divisa in tre liste che si chiamano Requestable, Returnable e Restrictable che rispettivamente indicano i dati che possono essere richiesti al nodo, i dati che possono essere restituiti dal nodo e quello che possono essere utilizzati come filtro per la richiesta.

Poiché tutte queste voci devono essere coerenti per tutti i nodi è stato definito un dictionary che elenca tutte quelle possibili con una descrizione, il tipo di dato. Il dictionary è consultabile all'indirizzo `http://dictionary.vamdc.eu/` e la cui documentazione è disponibile all'indirizzo `http://www.vamdc.org/documents/dictionary_v12.07.pdf`.

4.1 I protocolli del progetto

Per il progetto sono stati definiti due protocolli che vengono utilizzati per definite le richieste che arrivano ai nodi e le risposte che questi devono restituire.

Il protocollo di comunicazione per le query si chiama VAMDC SQL Subset 1 (VSS1) ed è essenzialmente simile ad SQL, studiato per fare in modo che il database possa essere indipendente dal linguaggio di query e che tutti i nodi possano ricevere ed interpretare le stesse query.

Il protocollo di risposta si chiama VAMDC-XSAMS[6] ed è una versione del protocollo XSAMS adattata alle esigenze del progetto VAMDC. XSAMS[7] (XML Schema for Atoms, Molecules and Solids) è uno standard internazionale IAEA. VAMDC-XSAMS è stato proposto anch'esso come standard all'IAEA. Di seguito un immagine dello schema:

4.2 TAP services

Il TAP (Table Access Protocol)[8] è uno standard definito dal Virtual Observatory come web service. Questo definisce la struttura dell'URL del nodo relativamente al servizio di query.

4.3 XSAMS Generator

Ogni nodo ha un modulo per la costruzione della risposta ad una query di interrogazione. Questa risposta deve rispettare le specifiche del protocollo VAMDC-XSAMS. Questo modulo è effettivamente l'unica parte di codice propriamente detto che è indispensabile sviluppare per adattare il software generico per un nodo VAMDC rilasciato, e mantenuto, dal progetto VAMDC, ad ogni specifico database [9].

5 Il Nodo VAMDC dell'Osservatorio Astronomico di Cagliari

Il Nodo è stato adattato, in una prima fase, sulla base del progetto ivh/VAMDC-VALD e in una fase successiva sul progetto VAMDC/NodeSoftware che è una fork del precedente progetto.

I files del nodo sono stati posizionati nella directory `/var/www/wsgi/VAMDC`

I files che sono stati modificati per le esigenze specifiche del Osservatorio Astronomico di Cagliari sono nella directory `/var/www/wsgi/VAMDC/oacagliari/`

In particolare nelle successive sezioni verranno illustrati e commentati i files che sono stati adattati

5.1 `oacagliari/apache.conf`

In questo file è stata copiata la configurazione di Apache 2.4 utilizzata dal server

```
<VirtualHost *:80>
        ServerName vamdc-pah.oa-cagliari.inaf.it
        ServerAlias vamdc-pah vamdc-pah.dsf.unica.it
        ServerAdmin webmaster@jansky.oa-cagliari.inaf.it

        # Make compression default
        AddOutputFilterByType DEFLATE application/xml
```

[6] `http://vamdc.eu/documents/standards/dataModel/vamdcxsams/index.html`
[7] `https://www-amdis.iaea.org/xml/about.html`
[8] `http://www.ivoa.net/documents/TAP/`
[9] `http://vamdc-nodesoftware.readthedocs.org/en/latest/index.html`

```
    AddOutputFilterByType DEFLATE text/xml
    AddOutputFilterByType DEFLATE application/x-votable+xml

    Alias /robots.txt /var/www/wsgi/static/robots.txt
    Alias /favicon.ico /var/www/wsgi/static/favicon.ico

    AliasMatch /([^/]*\.css) /var/www/wsgi/VAMDC/static/
styles/\$1
    AliasMatch /([^/]*\.xsl) /var/www/wsgi/VAMDC/static/xsl/\
$1
    AliasMatch /([^/]*\.xsd) /var/www/wsgi/VAMDC/static/xsd/\
$1

    Alias /media/ /var/www/wsgi/media/
    Alias /static/ /var/www/wsgi/static/

    <Directory /var/www/wsgi/static>
            Order deny,allow
            Require all granted
    </Directory>

    <Directory /var/www/wsgi/media>
            Order deny,allow
            Require all granted
    </Directory>

    WSGIScriptAlias / /var/www/wsgi/VAMDC/nodes/oacagliari/
django-vamdc.wsgi
    WSGIDaemonProcess vamdc_pah display-name=%{GROUP}
    WSGIProcessGroup vamdc_pah

    <Directory /var/www/wsgi/VAMDC/nodes/oacagliari>
                    Require all granted
    </Directory>
    <Directory /var/www/wsgi/VAMDC/nodes/oacagliari>
            Require all granted
    </Directory>

    #
    # Directives to allow use of AWStats as a CGI
    #
    Alias /awstatsclasses/ /var/www/wsgi/awstats/wwwroot/
classes/
    Alias /awstatscss/ /var/www/wsgi/awstats/wwwroot/css/
    Alias /awstats-icon/ /var/www/wsgi/awstats/wwwroot/icon/
    Alias /icon/ /var/www/wsgi/awstats/wwwroot/icon/
    ScriptAlias /awstats/ /var/www/wsgi/awstats/wwwroot/cgi-
bin/

    #
```

```
        # This is to permit URL access to scripts/files in
    AWStats directory.
        #
        #Alias /awstats "/var/www/wsgi/awstats/wwwroot/cgi-bin
        #<Directory "/var/www/wsgi/awstats/wwwroot/cgi-bin">
        # AddHandler cgi-script cgi pl
        # Options ExecCGI
        #</Directory>
        <Directory /var/www/wsgi/awstats/wwwroot/cgi-bin/>
            Options ExecCGI
            DirectoryIndex awstats.pl
            AllowOverride All
            Order allow,deny
            Require all granted
        </Directory>


        <Directory /var/www/wsgi/awstats/wwwroot/icon/>
          Options Indexes
          AllowOverride None
          Order allow,deny
          Require all granted
        </Directory>


        ErrorLog \${APACHE_LOG_DIR}/error.log

        # Possible values include: debug, info, notice, warn,
    error, crit,
        # alert, emerg.
        LogLevel debug

        CustomLog \${APACHE_LOG_DIR}/vamdcnode.access.log
    combined

</VirtualHost>
```

dove possiamo trovare la parte che imposta i parametri per far permettere ad Apache di gestire la web application del nodo scritta in Django tramite il WSGI (Web Server Gateway Interface).

```
        WSGIScriptAlias / /var/www/wsgi/VAMDC/nodes/oacagliari/
    django-vamdc.wsgi
        WSGIDaemonProcess vamdc_pah display-name=%{GROUP}
        WSGIProcessGroup vamdc_pah

        <Directory /var/www/wsgi/VAMDC/nodes/oacagliari>
                        Require all granted
        </Directory>
        <Directory /var/www/wsgi/VAMDC/nodes/oacagliari>
                Require all granted
```

```
        </Directory>
```

Mentre la parte seguente nel file è quella che attiva le funzionalità di statistica del nodo basate sull'analisi dei files di log di Apache tramite AWStats[11]

```
#
# Directives to allow use of AWStats as a CGI
#
Alias /awstatsclasses/ /var/www/wsgi/awstats/wwwroot/classes/
Alias /awstatscss/ /var/www/wsgi/awstats/wwwroot/css/
Alias /awstats-icon/ /var/www/wsgi/awstats/wwwroot/icon/
Alias /icon/ /var/www/wsgi/awstats/wwwroot/icon/
ScriptAlias /awstats/ /var/www/wsgi/awstats/wwwroot/cgi-bin/

#
# This is to permit URL access to scripts/files in AWStats
    directory.
#
#Alias /awstats "/var/www/wsgi/awstats/wwwroot/cgi-bin
#<Directory "/var/www/wsgi/awstats/wwwroot/cgi-bin">
# AddHandler cgi-script cgi pl
# Options ExecCGI
#</Directory>
<Directory /var/www/wsgi/awstats/wwwroot/cgi-bin/>
    Options ExecCGI
    DirectoryIndex awstats.pl
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>


<Directory /var/www/wsgi/awstats/wwwroot/icon/>
  Options Indexes
  AllowOverride None
  Order allow,deny
  Allow from all
</Directory>
```

5.2 oacagliari/django.wsgi

In questo file è stata copiata la configurazione del file di script utilizzata dal Web Server Gateway Interface per django

```
1 import os
2 import sys
3 sys.path.append('/var/www/wsgi/VAMDC')
4
5
6 os.environ['DJANGO_SETTINGS_MODULE'] = 'nodes.oacagliari.settings
    '
```

```
7
8  import django.core.handlers.wsgi
9  application = django.core.handlers.wsgi.WSGIHandler()
```

Le uniche personalizzazioni sono quelle relative al percorso in cui si trovano i files della web-application (riga 3) e le impostazioni del modulo che contiene i settings della web application (riga 6)

5.3 oacagliari/node/commonfunctions.py

Questo modulo è stato creato come libreria di alcune funzioni utilizzate dalla `queryfunction`. Sono funzioni per la gestione della voce del dictionary `RESTRICTABLE` AsOfDate, necessaria per garantire la riproducibilità delle queries.

```
import datetime

def datetimenow(returnobject = False):
    if returnobject:
        return datetime.datetime.now()
    else:
        return datetime.datetime.now().strftime("%Y-%m-%d␣%H:%M:%
    S")

def dataexpire(obj):
    old_obj = deepcopy(obj)
    old_obj.id = None
    old_obj.now()
    old_obj.save()
    obj.time_exp = datetimenow()
    obj.save()
    return old_obj

def checkexpired(obj, timecheck = None):
    result = False
    if obj.time_exp:
        if timecheck:
            if obj.time_exp < timecheck:
                result =  True
        else:
            if obj.time_exp < datetimenow(returnobject = True):
                result = True
    return result
```

La funzione `datetimenow` restituisce il datetime attuale e tramite l'argomento `returnobject` si può decidere il formato dell'output. La funzione `dataexpire` imposta il datetime di temine di validità dell'oggetto passato (`time_exp`) con il valore del datetime attuale. La funzione `checkexpired` restituisce vero o falso in base alla data di validità del dato passato (`time_exp`). Nel caso in cui venga passato anche l'argomento `timecheck` viene utilizzato questo per verificare `time_exp` altrimenti viene utilizzato il risultato della funzione `datetimenow` come temine di paragone.

5.4 oacagliari/node/dictionaries.py

In questo file sono state create le voci relative alle variabili RETURNABLE, RESTRICTABLE e REQUESTABLE.

REQUESTABLE: sono le variabili che possono essere richieste al nostro nodo (nel nostro caso non sono presenti variabili REQUESTABLE)

RESTRICTABLE: sono le variabili che possono essere utilizzate come filtro per le richieste

RETURNABLE: sono le variabili relative ai dati che potenzialmente possono essere restituiti dal nodo. Verranno restituiti solo i dati che sono stati specificati dalle REQUESTABLE o altrimenti tutti i dati possibili.

```
RETURNABLES = {
'NodeID':'OACagliari',
'MethodID':'Method.id',
'MethodCategory':'Method.category',
'MethodSourceRef':'Method.SourcesRef',
'MethodDescription':'Method.description',

#'AtomMass':'Atom.standard_atomic_weight',
#'AtomMassNumber':'Atom.atomic_mass',
#'AtomSymbol':'Atom.symbol',
#'MethodDescription': 'Method.category',
'MoleculeChemicalName':'Molecule.name',
'MoleculeInchi':'Molecule.inchi',
'MoleculeInchiKey':'Molecule.inchikey',
'MoleculeIonCharge':'Molecule.charge',
'MoleculeComment': 'Molecule.comments',
'MoleculeMolecularWeight':'Molecule.molweight()',
'MoleculeMolecularWeightUnit': u"amu",
#'MoleculeQnCase':'MoleQNs.',
'MoleculeSpeciesID':'Molecule.pk',
"MoleculeStateEnergyOrigin":"assuming␣zero␣at␣infinity",
'MoleculeStateEnergy':'MoleculeState.total_energy',
'MoleculeStateEnergyMethod':'MoleculeState.energymethod', #kkk
'MoleculeStateEnergyRef' : 'MoleculeState.StateEnergySourceRef',
    #kkk electronic state

'MoleculeStateEnergyUnit':u'au', #TO CHANGE
'MoleculeStateDescription': 'MoleculeState.description',
'MoleculeStateID':'MoleculeState.state_id',
'MoleculeOrdinaryStructuralFormula':'Molecule.
    OrdinaryStructuralFormula()',
'MoleculeStoichiometricFormula':'Molecule.formula',
'MoleculeStructure': "Molecule.molecularchemicalspecies",
'MoleculeStructureMethod': "MoleculeStructure.
    MoleculeStructureMethod", #kkk
'MoleculeStructureSourceRef': "MoleculeStructure.
    MoleculeStructureSourceRef", #kkk electronic state
```

```
#Normal Modes
'MoleculeNormalModes': 'Molecule.NomalModes', #'Molecule.',
'MoleculeNormalModesMethod' : 'NormalMode.NormalModesMethod',
'MoleculeNormalModesRef' : 'NormalMode.NormalModesSourceRef', #
    vibration analysis
'MoleculeNormalModeElectronicState': 'NormalMode.electronicstate'
    , #'Molecule.',
'MoleculeNormalModePointGroupSymmetry': 'NormalMode.
    pointgroupsymmetry', #'Molecule.',
'MoleculeNormalModeID' : 'NormalMode.normalmodeidtype',
'MoleculeNormalModeHarmonicFrequency' : 'NormalMode.
    HarmonicFrequency',#[0].HarmonicFrequency
'MoleculeNormalModeHarmonicFrequencyUnit' : u"MHz",
'MoleculeNormalModeIntensity' : 'NormalMode.intensity',
'MoleculeNormalModeIntensityUnit' : u"km/mol",
'MoleculeNormalModeDisplacementVectorX3' : 'NormalMode.
    displacementvectorsx',
'MoleculeNormalModeDisplacementVectorY3' : 'NormalMode.
    displacementvectorsy',
'MoleculeNormalModeDisplacementVectorZ3' : 'NormalMode.
    displacementvectorsz',
'MoleculeNormalModeDisplacementVectorRef' : 'NormalMode.
    displacementvectorselementref',
'MoleculeNormalModeDisplacementVectorsUnit' : '1/cm',

#'Sources' : 'Source' ,
#'SourceAuthors':'Source.Authors',
#'SourceCategory':'Source.category',
'SourceID':'Source.bib_id',
'SourceAuthorName':'Source.author',
'SourceCategory':'Source.category',
'SourcePageBegin':'Source.pages',
'SourcePageEnd':'Source.pages',
'SourceName':'Source.journal',
'SourceTitle':'Source.title',
'SourceURI':'Source.url',
'SourceVolume':'Source.volume',
'SourceYear':'Source.year',

}



RESTRICTABLES = {
'AsOfDate':'time_stamp',
#'AtomMass':'elementspecies__element__standard_atomic_weight',
#'AtomMassNumber':'elementspecies__element__atomic_mass',
#'AtomSymbol':'elementspecies__element__symbol',
'MoleculeChemicalName':'name',
'Inchi':'inchi',
```

```
'InchiKey':'inchikey',
'IonCharge':'charge',
'MoleculeStateEnergy':'electronicstates__total_energy',
'MoleculeStateID':'electronicstates__state_id',
'MoleculeStoichiometricFormula':'formula',
#'MoleculeMolecularWeight':'totalmolweight',
'MoleculeNormalModeHarmonicFrequency' : '
    electronicstates__vibrationalanalysesharmonic__tabulatedvibrations__frequency
    ',
'MoleculeNormalModeIntensity' : '
    electronicstates__vibrationalanalysesharmonic__tabulatedvibrations__ir_intensity
    ',
}



# Do not edit or remove these three lines
#from vamdctap.caselessdict import CaselessDict
#RETURNABLES = CaselessDict(RETURNABLES)
#RESTRICTABLES = CaselessDict(RESTRICTABLES)
```

5.5 oacagliari/node/models.py

In questo file sono stati specificati gli Object-Relational Mapping (ORM) che corrispondono alle tabelle e alle colonne del database relazionale del nodo.

È possibile trovare il contenuto del file in appendice

5.6 oacagliari/node/queryfunc.py

In questo file è stata scritta la funzione `setupResults` che tramite il dictionary che verrà passato al VAMDC TAP (Table Access Protocol) Generator. Questo si occuperà di costruire il file XML che sarà la risposta del nodo alle richieste.

```
# -*- coding: utf-8 -*-
#
# This module (which must have the name queryfunc.py) is
    responsible
# for converting incoming queries to a database query understood
    by
# this particular node's database schema.
#
# This module must contain a function setupResults, taking a sql
    object
# as its only argument.
#

# library imports

import sys
```

```
import datetime
from itertools import chain
from django.conf import settings
from vamdctap.sqlparse import where2q, splitWhere,
    mergeQwithLogic, restriction2Q


import dictionaries

import models # this imports models.py from the same directory as
     this file
from SOAPpy.wstools.WSDLTools import Element
from django.db.models import Q


def LOG(s):
    "Simple␣logger␣function"
    if settings.DEBUG: print >> sys.stderr, s

#----------------------------
# Helper functions (called from setupResults)
#----------------------------
def MoleculeNormalModeHarmonicFrequency_MHz2cm(op, foo):
    """
␣␣␣␣Making␣the␣conversion␣from␣MHz␣to␣cm-1.
␣␣␣␣"""

    opp = op

    try:
        foop = float(foo[0].replace("'", "").replace('"', ''))
    except (ValueError, TypeError):
        print 'failed␣to␣convert␣%s␣to␣float' % foo
        return None
    except (IndexError):
        print 'no␣argument␣to␣MoleculeNormalModeHarmonicFrequency
    ␣restrictable'
        return None
    if foop != 0.:
        # MHz -> cm-1
        foop =  foop / 29979.2458
    q = ['MoleculeNormalModeHarmonicFrequency', opp, str(foop)]
    return q

def CreateDateLowerBound(r, op, foo):
    """
␣␣␣␣This␣function␣convert␣the␣equal␣operator␣with␣lte
␣␣␣␣"""

    opp = op
```

```
    if op == "=" or op == ">":
        opp= "<"
    q = [r, opp, foo[0]]
    return q
#------------------------------
# Main function
#------------------------------



def builRef(id_ref):
    b = models.Bibliography.alive_objects.get(pk = id_ref)
    return b.bibtextoref()


def setupResults(sql, limit=1000):

    #This function is always called by the software.

    # log the incoming query
    LOG(sql)
    result_sources = [] #Sources related to results
    methods = [] #Methods related to results
    nmMethod = None
    # convert the incoming sql to a correct django query syntax
    object
    # based on the RESTRICTABLES dictionary in dictionaries.py
    # (where2q is a helper function to do this for us).


    q = where2q(sql.where, dictionaries.RESTRICTABLES)
    try:
        q = eval(q) # test queryset syntax validity
    except Exception,e:
        return {}
    # ... we parse the query into its restrictables and logic:
    if not sql.where:
        return {}
    logic, rs, count = splitWhere(sql.where)
    indexAsOfDate = []
    i_counter = 0
    AsOfDateValues = []
    for i in rs:
        r, op, foo = rs[i][0], rs[i][1], rs[i][2:]
        if r == 'MoleculeNormalModeHarmonicFrequency':
            #Convert MoleculeNormalModeHarmonicFrequency
            # from MHz to cm-1:
            rs[i] = MoleculeNormalModeHarmonicFrequency_MHz2cm(op
    , foo)
        if r == "AsOfDate":
```

```
        #create date lower bound
        indexAsOfDate.append(i_counter)
        rs[i] = CreateDateLowerBound(r, op, foo)
        AsOfDateValues.append(datetime.datetime(*[eval(d) for
 d in foo[0].replace('"', "").split("-")]))
    i_counter += 1
# now rebuild the query as a dictionary of QTypes ...
qdict = {}
for i, r in rs.items():
    q = restriction2Q(r)
    qdict[i] = q
# ... and merge them to a single query object
q = mergeQwithLogic(qdict, logic)


# We build a queryset of database matches on the Transition
model
# since through this model (in our example) we are be able to
# reach all other models. Note that a queryset is actually
not yet
# hitting the database, making it very efficient.


if len(indexAsOfDate)>0:
    AsOfDateValue = AsOfDateValues[0]
    #moleculars pecies = models.MolecularSpecies.alive_objects
.filter(q).distinct()
#else:
    #for i in indexAsOfDate:
    #    q.children[eval(i)] = (q.children[eval(i)][0],
AsOfDateValue[i])

    #q_upperbound = Q(time_exp__gte = AsOfDateValue)| Q(
time_exp__isnull=True)
    #moleculars pecies = models.MolecularSpecies.objects.
filter(q & q_upperbound).distinct()
    #select all objects born before AsOfDate
    molecularspecies = models.MolecularSpecies.objects.filter
(q).distinct()

    notexpired = []
    for ms in molecularspecies:
        if not ms.expired(timecheck = AsOfDateValue):
            notexpired.append(ms.pk)
    #remove all objects not dead after AsOfDate
    molecularspecies = molecularspecies.filter(pk__in =
notexpired)
else:
    #select only objects that are alive now
```

```
    molecularspecies = models.MolecularSpecies.alive_objects.
filter(q).distinct()
    AsOfDateValue = None

# count the number of matches, make a simple trunkation if
there are
# too many (record the coverage in the returned header)
nmolecularspecies=molecularspecies.count()
if limit < nmolecularspecies :
    nmolecularspecies = molecularspecies[:limit]
    percentage='%.1f' % (float(limit) / nmolecularspecies *
100)
else:
    percentage=None

nspecies = molecularspecies.count()
nstates = 0

for molecular_specie in molecularspecies:
    es_qs = molecular_specie.electronicstates_set.all()
    notexpired = []
    for es in es_qs:
        if not es.expired(timecheck = AsOfDateValue):
            notexpired.append(es.pk)
    molecular_specie.States = es_qs.filter(pk__in =
notexpired)
    min_energy = 0
    min_energy = min
    normalmodelist = []
    NormalModeSourceRef = []
    MoleculeStructureSourceRef = []
    for state in molecular_specie.States:
        #search minimum total energy
        state.StateEnergySourceRef = []
        state.energymethod = state.task.pk
        if not (state.energymethod in [m.id for m in methods
]):
            new_method = models.Method(state.energymethod,
state.task.returnmethoddescriptionandbib(), result_sources)
            result_sources += new_method.Sources
            methods.append(new_method)

        for ref in state.bibliographies.all():
            if ref.bibtex:
                if not ref.expired(timecheck = AsOfDateValue)
:
                    if not (ref in result_sources):
                        result_sources.append(ref)
                    state.StateEnergySourceRef.append(ref.
bib_id)
```

```
        if min([min_energy, state.total_energy]) == state.
total_energy:
            #we need to list all molecule structure
            MoleculeStructureMethod = state.energymethod
            MoleculeStructureSourceRef = state.
StateEnergySourceRef
            MoleculeStructure = state.geom.returncmlstructure
(MoleculeStructureSourceRef)
        notexpired = []
        state_va = state.vibrationalanalysesharmonic_set.all
()
        for va in state_va:
            if not va.expired(timecheck = AsOfDateValue):
                notexpired.append(va.pk)
        vibration_analyses_harmonic = state_va.filter(pk__in
= notexpired)
        elementlist = state.geom.returnelementslist()

        if len(vibration_analyses_harmonic) > 0:
            vah = vibration_analyses_harmonic[0]
            for vref in vah.bibliographies.all():
                if vref.bibtex:
                    if not vref.expired(timecheck =
AsOfDateValue):
                        if not (vref in result_sources):
                            result_sources.append(vref)
                        NormalModeSourceRef.append(vref.
bib_id)
            nmMethod = vah.task.pk
            if not (nmMethod in [m.id for m in methods]):
                new_method = models.Method(nmMethod, vah.task
.returnmethoddescriptionandbib(), result_sources)
                result_sources += new_method.Sources
                methods.append(new_method)

            for tab in vah.tabulatedvibrations_set.all():
                normalmode = models.NormalMode(tab.pk, tab.
frequency, tab.ir_intensity, tab.sym_type, tab.eigenvectors,
state.state_id, elementlist, nmMethod, NormalModeSourceRef)
                normalmodelist.append(normalmode)

    molecular_specie.moleculararchemicalspecies = models.
MolecularChemicalSpecies(MoleculeStructure,
MoleculeStructureMethod, MoleculeStructureSourceRef, None,
nmMethod, NormalModeSourceRef)
    if normalmodelist:
        molecular_specie.NormalModes = normalmodelist
    nstates += molecular_specie.States.count()
    molecular_specie.comments = str(molecular_specie.comments
)
```

```
# Through the transition-matches, use our helper functions to
 extract
# all the relevant database data for our query.

# Create the header with some useful info. The key names here
 are
# standardized and shouldn't be changed.
nsources = len(result_sources)
nmolecules = len(molecularspecies)


if molecularspecies:
    size_estimate='%.2f'%(nmolecules*0.0014 + 0.01)
else: size_estimate='0.00'

headerinfo={\
        'TRUNCATED':percentage,
        'COUNT-STATES':nstates,
        'COUNT-MOLECULES': nmolecules,
        'COUNT-SPECIES':nspecies,
        'APPROX-SIZE':size_estimate,
        }

# Return the data. The keynames are standardized.
return {"Molecules" : molecularspecies,
        "Methods" : methods,
        'Sources':result_sources,
        'HeaderInfo':headerinfo,
       }
```

La funzione principale del modulo è `setupResults(sql, limit=1000)` che imposta `molecularspecies` con la lista delle molecole che corrispondono ai criteri di ricerca e per ogni molecola associa gli stati elettronici e a questi i dati delle analisi vibrazionali. Questi dati vengono aggiunti seguendo una nomenclatura impostata dal dictionary `RETURNABLES` visto in precedenza.

5.7 `oacagliari/settings.py`

In questo file sono state inserite le informazioni relative all'accesso al database (nome utente, password, IP e porta, motore di database (RDBMS)).

6 PAH database web frontend

Per permettere una semplice navigazione dei dati presenti nel database è stata sviluppata un'interfaccia web che presenta tutti i risultati dei calcoli divisi per molecole.

Il sito web è disponibile all'indirizzo `http://qchitool-pah-dev.oa-cagliari.inaf.it/` ed è stato illustrato in [12]. Le informazioni disponibili per specie molecolari presenti nel database `pah_data` includono dati energetici e strutturali, analisi vibrazionali e spettri di assorbimento fotoelettrico.

Inoltre nel sito sono state integrate le funzionalità di inserimento di nuovi dati nel database come illustrate in precedenza. Questo frontend è stato sviluppato come web application tramite il framework Django e permette tramite Jmolapplet la visualizzazione delle molecole e l'interazione con il loro modello 3D.

In Fig. 6 il diagramma di flusso da parte dell'utente.

7 Conclusioni e sviluppi futuri

Il database presenta per ora una piccola parte dei dati risultato dei calcoli effettuali dal gruppo di astrochimica dell'Osservatorio Astronomico di Cagliari. Questo perché il processo di inserimento richiede particolare attenzione del completare le informazioni che non possono essere estratte in maniera automatica dal QChITool.

Il QChITool nella sua versione attuale permette l'inserimento dei dati esclusivamente di NwChem, le parti del codice necessarie per l'inserimento nel database dei calcoli di altri quantum chemistry software sono in sviluppo.

Il Nodo VAMDC dell'Osservatorio viene via via aggiornato alle versioni delle successive release sviluppate dal consorzio. In queste nuove release sono state abilitate delle nuove funzionalità che hanno richiesto in alcuni casi delle integrazioni nel codici.

Il web frontend rende disponibile l'accesso ai dati presenti nel database e nel suo sviluppo futuro prevederà alcune funzioni di ricerca e filtro.

8 Appendice

8.1 Codifica di funzioni e variabili

Per semplificare le stringhe che identificano funzioni e variabili del codice è stata utilizzata la seguente tabella di prefissi:

- `[1] qchitool.parser.nwchem.check_import_object`
- `[2] qchitool.parser.impnwc`
- `[3] qchitool.parser.nwchem.commons`
- `[4] qchitool.parser.nwchem.nwc_sections.nwc_sec_commons`
- `[5] qchitool.parser.commonsobjects`
- `[6] qchitool.tools.modeltools`
- `[7] qchitool.parser.nwchem.nwc_sections.nwc_mapping`

- [8] qchitool.parser.nwchem.nwc_sections

```
# This is an auto-generated Django model module.
# You'll have to do the following manually to clean this up:
#     * Rearrange models' order
#     * Make sure each model has one field with primary_key=True
# Feel free to rename the models, but don't rename db_table
    values or field names.
#
# Also note: You'll have to insert the output of 'django-admin.py
     sqlcustom [appname]'
# into your database.

from django.db import models
from decimal import *
import commonfunctions as cf
import re

import dictionaries

from vamdctap.bibtextools import *
from vamdctap.generators import makeSourceRefs
import os
if os.getenv('ENV_USER_TZ', None):
    TIME_ZONE = os.getenv('ENV_USER_TZ') # changed to UTC

class GenericModelNotExpiredManager(models.Manager):
    def get_query_set(self):
        return self.get_alive_query_set()
    def get_alive_query_set(self, timecheck = None):
        wanted_items = set()
        for item in super(GenericModelNotExpiredManager, self).
    get_query_set().all():
            if not item.expired(timecheck):
                wanted_items.add(item.pk)
        return super(GenericModelNotExpiredManager, self).
    get_query_set().filter(pk__in = wanted_items)

class Author(object):
    def __init__(self, Name, Address):
        self.Name = Name
        self.Address = Address

class BibRef(object):
    # simple dummy object to define a Method
    def __init__(self, SourceID, Category, SourceName, Year,
    Authors, Title):
```

```
        self.SourceID = SourceID
        self.Category = Category
        self.SourceName = SourceName
        self.Year = Year
        self.Authors = Authors
        self.Title = Title

class Method(object):
    # simple dummy object to define a Method
    def __init__(self, mid, description, Sources):
        self.id = mid
        self.category = 'theory'
        self.description, tmpbib = description
        self.Sources = []
        for bib in tmpbib:
            if bib.bibtex:
                if not (bib in Sources):
                    self.Sources.append(bib)
        self.SourcesRef = [b.pk for b in self.Sources]


class NormalMode(models.Model):
    def __init__(self, normalmode_id, frequency, intensity,
    sym_type, displacementvectors, electronicstate, elements,
    Methods, SourceRefs):
        #electronicStateRef
        self.pointgroupsymmetry = sym_type
        self.normalmodeidtype = normalmode_id #NormalModeIDType,
    defining unique identifier for this mode, to be referenced
        self.HarmonicFrequency = str(frequency * Decimal('
    29979.2458'))
        self.intensity = intensity
        self.electronicstate = electronicstate
        self.NormalModesMethod = Methods
        self.NormalModesSourceRef = SourceRefs
        if displacementvectors:
            self.displacementvectors = eval(displacementvectors)
            self.displacementvectorsx = []
            self.displacementvectorsy = []
            self.displacementvectorsz = []
            self.displacementvectorselementref = []
            for index in range(len(elements)): #XXX
                self.displacementvectorselementref.append(
    elements[index])
                self.displacementvectorsx.append(self.
    displacementvectors[0 + index * 3 ])
                self.displacementvectorsy.append(self.
    displacementvectors[1 + index * 3 ])
                self.displacementvectorsz.append(self.
    displacementvectors[2 + index * 3 ])
```

```
        else:
            self.displacementvectors = None
    def returnXML(self, elements, method, RefSource, NodeName):

        result = '<NormalMode id="V' + NodeName + "-" + str(self.
    normalmodeidtype) + '"'
        if self.pointgroupsymmetry:
            result += ' pointGroupSymmetry="' + self.
    pointgroupsymmetry + '"'
        if method:
            result += ' methodRef="M' + NodeName + "-" + str(
    method) + '"'
        result += '>\n'
        #result += returnXMLSource(RefSource, NodeName)
        result += '<HarmonicFrequency>\n'
        result += '<Value units="MHz">' + str(self.
    harmonicfrequency * Decimal('29979.2458')) + '</Value>\n'
        #result += '<Accuracy>1</Accuracy>\n'
        result += '</HarmonicFrequency>\n'
        result += '<Intensity>\n'
        result += '<Value units="km/mol">' + str(self.intensity)
    + '</Value>\n'
        result += '</Intensity>\n'
        if self.displacementvectors:
            result += '<DisplacementVectors units="1/cm">\n'
            for index in range(len(elements)): #XXX
                result += '<Vector ref="' + elements[index] + '"'
                result += ' x3="' + self.displacementvectors[0 +
    index * 3 ] + '"'
                result += ' y3="' + self.displacementvectors[1 +
    index * 3 ] + '"'
                result += ' z3="' + self.displacementvectors[2 +
    index * 3 ] + '"/>\n'
            result += '</DisplacementVectors>\n'

        result +="</NormalMode>\n"
        return result


class MolecularChemicalSpecies():
    def __init__(self, moleculestructure, MoleculeStructureMethod
    , MoleculeStructureSourceRef, normalmodes, NormalModesMethod,
     NormalModesSourceRef):
        #self.moleculestructure = moleculestructure.replace("<
    molecule>", '<cml xmlns="http://www.xml-cml.org/schema" xsi:
    schemaLocation="http://www.xml-cml.org/schema ../../schema.
    xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">')
    .replace("</molecule>", "</cml>")
        self.moleculestructure = moleculestructure
        self.MoleculeStructureMethod = MoleculeStructureMethod
```

```
        #self.normalmodes = returnXMLSource(NormalModesSourceRef,
     NodeName) + normalmodes
        #self.normalmodes = normalmodes
        #self.NormalModesMethod = NormalModesMethod
        #self.NormalModesSourceRef = NormalModesSourceRef
        self.MoleculeStructureSourceRef =
    MoleculeStructureSourceRef

    def CML(self):
        return self.moleculestructure.replace("<molecule>", "").
    replace("</molecule>", "").replace("atom␣", "cml:atom␣").
    replace("atomA", "cml:atomA").replace("bond", "cml:bond")


class Bibliography(models.Model):
    bib_id = models.AutoField(primary_key=True)
    type = models.ForeignKey('Reftype')
    series = models.ForeignKey('PublicationSeries')
    authors = models.ManyToManyField('Authors') # , through = "
    AuthorGroups"
    editors = models.ManyToManyField('Editors') # , through = "
    EditorsGroups"
    publisher = models.ForeignKey('Publishers')
    title = models.TextField(blank=True)
    volume = models.IntegerField(null=True, blank=True)
    doi = models.TextField(blank=True)
    page_begin = models.CharField(max_length = 45, blank=True)
    page_end = models.CharField(max_length = 45,blank=True)
    uri = models.TextField(blank=True)
    city = models.CharField(max_length = 45, blank=True)
    version = models.CharField(max_length = 45, blank=True)
    source_name = models.TextField(blank=True)
    date = models.DateField(null=True, blank=True)
    reference = models.TextField(blank=True)
    bibtex = models.TextField(blank=True)
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'bibliography'
    def authorlist(self):
        result = u""
        for a in self.authors.all():
            result += a.name + u";␣"
        return result
    def __unicode__(self):
        return self.title + self.authorlist()
    def XML(self):
```

```
        """
␣␣␣␣␣␣␣␣This␣function␣replace␣the␣source␣ID␣with␣the␣database␣ID␣
    Bibliography␣record
␣␣␣␣␣␣␣␣and␣return␣the␣hard␣XML␣code␣by␣the␣function␣BibTeX2XML
␣␣␣␣␣␣␣␣"""
        try: NODEID = dictionaries.RETURNABLES['NodeID']
        except: NODEID = 'PleaseFillTheNodeID'
        if self.bibtex:
            r = r"sourceID=\"B"+␣NODEID␣+␣r"-.*\""
            newidvalue = u'sourceID="B'+ NODEID + u"-" + str(self
    .bib_id) + u'"'
            result = re.sub(r, newidvalue,  BibTeX2XML( self.
    bibtex ))
            return unicode(result)
    def bibtextoref(self):
        if self.bibtex:
            from pybtex.database.input import bibtex
            parser = bibtex.Parser()
            bib_data = parser.parse_stream(self.bibtex)
            if "title" in bib_data.entries[bib_data.entries.keys
    (0)].fields:
                tmp_title = bib_data.entries[bib_data.entries.
    keys(0)].fields['title']
            else:
                tmp_title = None
            if "pages" in bib_data.entries[bib_data.entries.keys
    (0)].fields:
                tmp_page_begin, tmp_page_end  = re.split(r"[,-]",
     bib_data.entries[bib_data.entries.keys(0)].fields['pages'])
            if "volume" in bib_data.entries[bib_data.entries.keys
    (0)].fields:
                tmp_volume = bib_data.entries[bib_data.entries.
    keys(0)].fields['volume']
            if "doi" in bib_data.entries[bib_data.entries.keys(0)
    ].fields:
                tmp_doi = bib_data.entries[bib_data.entries.keys
    (0)].fields['doi']
            return BibRef(self.bib_id, "book", "SourceName", "
    2011", [Author("Author01", None), Author("Author02", None)],
    tmp_title)

    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
```

```
    alive_objects = GenericModelNotExpiredManager()


class Authors(models.Model):
    author_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length = 200, blank=True)
    address = models.TextField(blank=True)
    email = models.EmailField(blank=True, verbose_name = "e-mail"
    )
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'authors'
    def __unicode__(self):
        return self.name
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()



class BasisSets(models.Model):
    basisset_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length = 45, blank=True, null=
    True,)
    description = models.TextField(blank=True, null=True,)
    bibliographies = models.ManyToManyField('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True)
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'basissets'
    def __unicode__(self):
        returnstring = ""
        if self.name:
            returnstring += self.name
        else:
            returnstring += "NO␣NAME"
        if self.description:
            returnstring += "␣-␣" + self.description
```

```
        return returnstring


class BasisSetsBibliographies(models.Model):
    id = models.AutoField(primary_key=True)
    basissets = models.ForeignKey('BasisSets')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'basissets_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()



class BibliographyAuthors(models.Model):
    id = models.AutoField(primary_key=True)
    authors = models.ForeignKey('Authors')
    bibliographies = models.ForeignKey('Bibliography')
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'bibliography_authors'


class BibliographyEditors(models.Model):
    id = models.AutoField(primary_key=True)
    bibliographies = models.ForeignKey('Bibliography')
    editors = models.ForeignKey('Editors')
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'bibliography_editors'


class CalculationGroups(models.Model):
    calcgroup_id = models.AutoField(primary_key=True)
    comments = models.TextField(blank=True)
    class Meta:
```

```
        db_table = u'calculationgroups'


class ChemistryCodes(models.Model):
    code_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length = 200, null = True, blank=
    True)
    version = models.CharField(max_length = 45, null = True,
    blank=True)
    description = models.TextField( null = True, blank=True)
    bibliographies = models.ManyToManyField('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(blank=True, null = True)
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'chemistrycodes'
    def __unicode__(self):
        returnstring = ""
        if self.name:
            returnstring += self.name
        return self.name + "␣-␣" + self.version


class Calculations(models.Model):
    calc_id = models.AutoField(primary_key=True)
    code = models.ForeignKey('ChemistryCodes')
    input = models.TextField(blank=True)
    input_md5 = models.CharField(max_length = 45, blank=True)
    output = models.TextField(blank=True)
    output_md5 = models.CharField(max_length = 45, blank=True)
    other_output = models.TextField(blank=True)
    other_output_md5 = models.CharField(max_length = 45, blank=
    True)
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True)
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'calculations'

    def __unicode__(self):
        stringreturn = ""
        if self.time_stamp:
            stringreturn += "Entry␣date:␣" + self.time_stamp + "\
    n"
        if self.qual_index:
            stringreturn += "Quality␣index:␣" + self.qual_index +
     "\n"
        if self.code:
            stringreturn += "code:␣" + str(self.code) + "\n"
```

```
        if self.input_md5:
            stringreturn += "Input␣File␣md5:␣" + self.input_md5 +
     "\n"
        if self.output_md5:
            stringreturn += "Output␣File␣md5:␣" + self.output_md5
     + "\n"
        if self.other_output_md5:
            stringreturn += "Other␣Output␣File␣md5:␣" + self.
    other_output_md5 + "\n"
        if len(stringreturn) > 0:
            return stringreturn
        else:
            return "No␣Data"


class CalculationLists(models.Model):
    calc_list_id = models.AutoField(primary_key=True)
    calcgroup = models.ForeignKey('CalculationGroups')
    calc = models.ForeignKey('Calculations')
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'calculationlists'


class ChemistryCodesBibliographies(models.Model):
    id = models.AutoField(primary_key=True)
    chemistrycodes = models.ForeignKey('ChemistryCodes')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'chemistrycodes_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class DdResonances(models.Model):
    dd_id = models.AutoField(primary_key=True)
    vibanalisysanarmonic = models.ForeignKey('
    VibrationalAnalysesAnarmonic')
```

```
    vib_id1 = models.ForeignKey('TabulatedVibrations', db_column=
    'vib_id1', related_name='vib_id1')
    vib_id2 = models.ForeignKey('TabulatedVibrations', db_column=
    'vib_id2', related_name='vib_id2')
    k = models.FloatField(null=True, blank=True)
    class Meta:
        db_table = u'ddresonances'



class DipoleMoments(models.Model):
    dip_id = models.AutoField(primary_key=True)
    state = models.ForeignKey('ElectronicStates')
    task = models.ForeignKey('Tasks')
    mu_x = models.FloatField()
    mu_y = models.FloatField()
    mu_z = models.FloatField()
    bibliographies = models.ManyToManyField('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'dipolemoments'
    def __unicode__(self):
        return u"mu_x:␣" + str(self.mu_x) + u"␣mu_y:␣" + str(self
    .mu_y) + u"␣mu_z:␣" + str(self.mu_z)
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()



class DipoleMomentsBibliographies(models.Model):
    id = models.AutoField(primary_key= True)
    dipolemoments = models.ForeignKey('DipoleMoments')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
```

```
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'dipolemoments_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


#class DjangoAdminLog(models.Model):
#    id = models.IntegerField(primary_key=True)
#    action_time = models.DateTimeField()
#    user_id = models.IntegerField()
#    content_type_id = models.IntegerField(null=True, blank=True)
#    object_id = models.TextField(blank=True)
#    object_repr = models.CharField(max_length=600)
#    action_flag = models.IntegerField()
#    change_message = models.TextField()
#    class Meta:
#        db_table = u'django_admin_log'


#class DjangoContentType(models.Model):
#    id = models.IntegerField(primary_key=True)
#    name = models.CharField(max_length=300)
#    app_label = models.CharField(unique=True, max_length=255)
#    model = models.CharField(unique=True, max_length=255)
#    class Meta:
#        db_table = u'django_content_type'


#class DjangoSession(models.Model):
#    session_key = models.CharField(max_length=120, primary_key=
    True)
#    session_data = models.TextField()
#    expire_date = models.DateTimeField()
#    class Meta:
#        db_table = u'django_session'


#class DjangoSite(models.Model):
#    id = models.IntegerField(primary_key=True)
#    domain = models.CharField(max_length=300)
#    name = models.CharField(max_length=150)
```

```
#    class Meta:
#        db_table = u'django_site'


class Editors(models.Model):
    editor_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length = 45, blank=True)
    address = models.TextField(blank=True)
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'editors'
    def __unicode__(self):
        return self.name
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager() # The Dahl-
    specific manager.


class ElectronicStatesBibliographies(models.Model):
    id = models.AutoField(primary_key= True)
    electronicstates = models.ForeignKey('ElectronicStates')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'electronicstates_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
```

```
    alive_objects = GenericModelNotExpiredManager()


class ElectronicTransitions(models.Model):
    transition_id = models.AutoField(primary_key=True)
    low_state = models.ForeignKey('ElectronicStates',
    related_name = 'low_state')
    up_state = models.ForeignKey('ElectronicStates', related_name
     = 'up_state')
    task = models.ForeignKey('Tasks')
    energy = models.FloatField(null=True, blank=True)
    osc_strenght = models.FloatField(null=True, blank=True)
    mu_x = models.FloatField(null=True, blank=True)
    mu_y = models.FloatField(null=True, blank=True)
    mu_z = models.FloatField(null=True, blank=True)
    bibliographies = models.ManyToManyField('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'electronictransitions'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class ElectronicTransitionsBibliographies(models.Model):
    id = models.AutoField(primary_key= True)
    electronictransitions = models.ForeignKey('
    ElectronicTransitions')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'electronictransitions_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
```

```
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class Elements(models.Model):
    element_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length = 200, blank=True)
    symbol = models.CharField(max_length = 45, blank=True)
    atomic_number = models.IntegerField(null=True, blank=True)
    atomic_mass = models.IntegerField(null=True, blank=True)
    element_group = models.CharField(max_length = 45, blank=True)
    standard_atomic_weight = models.FloatField(null=True, blank=
    True)
    isotope_of = models.ForeignKey("self", null=True, blank=True)
    class Meta:
        db_table = u'elements'
    def __unicode__(self):
        return self.name + "(" + self.symbol + ",␣" + str(self.
    atomic_number) +  ",␣" + str(self.atomic_mass) + ")"
    def electron_number(self):
        return self.atomic_number


class Xcclasses(models.Model):
    xcclasses_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length = 200, blank=True)
    description = models.TextField(blank=True)
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'xcclasses'
    def __unicode__(self):
        if self.xcclasses_id:
            if self.name:
                return u"XC␣Class␣" + str(self.name)
            else:
                return u"XC␣Class␣" + str(self.xcclasses_id)
        else:
            return u"XC␣Class␣0"



class Geometryclasses(models.Model):
    geometryclass_id = models.AutoField(primary_key=True)
    geometry = models.TextField(blank=True, verbose_name = "
    Geometry␣(CML␣Format)")
```

```
    geometry_md5 = models.CharField(max_length = 45, blank=True)
    sym_group = models.TextField(blank=True, verbose_name = "
    Symmetry␣Group")
    sym_elements = models.TextField(blank=True, verbose_name = "
    Symmetry␣Elements")
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'geometryclasses'
    def __unicode__(self):
        if self.geometryclass_id:
            return u"Geometry␣Class␣" + str(self.geometryclass_id
    )
        else:
            return u"Geometry␣Class␣0"
    def calculate_md5(self):
        if self.geometry:
            self.geometry_md5 = returnmd5(re.split("[\r\n]+",
    unicode(self.geometry), 2)[2])
    def returncmlstructure(self):
        #this function return the geometry stored in self.
    geometry (cml format)
        return self.geometry


class FermiResonances(models.Model):
    fermi_id = models.AutoField(primary_key=True)
    vibanalisysanarmonic = models.ForeignKey('
    VibrationalAnalysesAnarmonic')
    vib_id1 = models.ForeignKey('TabulatedVibrations', db_column=
    'vib_id1', related_name='vib_id3')
    vib_id2 = models.ForeignKey('TabulatedVibrations', db_column=
    'vib_id2', related_name='vib_id4')
    vib_id3 = models.ForeignKey('TabulatedVibrations', db_column=
    'vib_id3', related_name='vib_id5')
    fi = models.FloatField(null=True, blank=True)
    class Meta:
        db_table = u'fermiresonances'


class Geometries(models.Model):
    geom_id = models.AutoField(primary_key=True)
    geometry = models.TextField(blank=True, verbose_name = "
    Geometry␣(CML␣Format)")
    geometry_md5 = models.CharField(max_length = 45, blank=True)
    sym_group = models.TextField(blank=True, verbose_name = "
    Symmetry␣Group")
    sym_elements = models.TextField(blank=True, verbose_name = "
    Symmetry␣Elements")
    geometryclass = models.ForeignKey('Geometryclasses')
```

```
geometryclass_rotationalmatrix = models.TextField(blank=True,
 verbose_name = "Rotational␣Matrix")
geometryclass_errors = models.TextField(blank=True,
verbose_name = "Geometry␣Error")
time_stamp = models.DateTimeField(null=True, blank=True)
qual_index = models.IntegerField(null=True, blank=True,
verbose_name = "Quality␣Index")
comments = models.TextField(blank=True)
time_exp = models.DateTimeField(null=True, blank=True)
class Meta:
    db_table = u'geometries'
def __unicode__(self):
    result = u""
    electronicstates = ElectronicStates.alive_objects.filter(
geom = self.geom_id, is_minimum = True )
    if len(electronicstates) > 0:
        if self.geometry_md5:
            result += u"MD5:␣" + self.geometry_md5 + u",␣"
        if self.sym_group:
            result += u"Symmetry␣Group:" + self.sym_group + u
",␣"
        result += unicode(electronicstates[0].species)

    else:
        if self.geometry_md5:
            result =  u"MD5:␣" + self.geometry_md5
        else:
            result =  self.geometry[0:20] + u"..."
    return result

def calculate_md5(self):
    if self.geometry:
        self.geometry_md5 = returnmd5(re.split("[\r\n]+",
unicode(self.geometry), 2)[2])
def returncmlstructure(self, SourceRefIDs = None):
    #this function return the geometry stored in self.
geometry (cml format)
    if SourceRefIDs:
        return makeSourceRefs(SourceRefIDs) + self.geometry
    else:
        return self.geometry
def returnelementslist(self):
    #this function return the list of elements
    result = []
    atoms = re.findall('id=".*"␣element', self.geometry)
    for row in atoms:
        s = row.split('"')
        result.append(s[1])
        #result.append(s[3] + s[1].replace("a", None))
    return result
```

```
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()



class IonisationEnergies(models.Model):
    ion_id = models.AutoField(primary_key=True)
    start_state = models.ForeignKey('ElectronicStates',
    related_name = 'start_state')
    ion_state = models.ForeignKey('ElectronicStates',
    related_name = 'ion_state')
    iontype = models.ForeignKey('IonisationTypes')
    energy = models.FloatField(null=True, blank=True)
    bibliographies = models.ManyToManyField('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'ionisationenergies'
    def __unicode__(self):
        result = u""
        if self.start_state.species.inchi:
            result += u"from␣" + self.start_state.species.inchi
        if self.ion_state.species.inchi:
            result += u"␣to␣" + self.ion_state.species.inchi
        if self.iontype:
            result += u",␣Type␣" + self.iontype.description
        if self.energy:
            result += u",␣Energy␣" + str(self.energy)
        return result
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()
```

```
class IonisationEnergiesBibliographies(models.Model):
    id = models.AutoField(primary_key= True)
    ionisationenergies = models.ForeignKey('IonisationEnergies')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'ionisationenergies_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()



class IonisationTypes(models.Model):
    iontype_id = models.AutoField(primary_key=True)
    description = models.TextField(blank=True)
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'ionisationtypes'
    def __unicode__(self):
        result = u""
        if self.description:
            result = u"IonType:␣" + self.description
        else:
            if self.iontype_id:
                result =  u"IonType␣ID:␣" + str(self.iontype_id)
            else:
                result =  u"IonType␣ID:␣0"
        return result




class MolecularSpecies(models.Model):
    species_id = models.AutoField(primary_key=True)
    name = models.TextField(null=True, blank=True)
```

```
formula = models.TextField(null=True, blank=True,
verbose_name = "Stoichiometryc␣Formula")
inchi = models.TextField(null=True, blank=True)
inchikey = models.CharField(max_length = 45, blank=True)
aromatic_cycles = models.IntegerField(null=True, blank=True)
charge = models.IntegerField(null=True, blank=True)
time_stamp = models.DateTimeField(null=True, blank=True)
qual_index = models.IntegerField(null=True, blank=True,
verbose_name = "Quality␣Index")
comments = models.TextField(blank=True)
isotopologue_of = models.ForeignKey("self", blank=True, null=
True)
time_exp = models.DateTimeField(null=True, blank=True)

def _get_molweight(self): return self.molweight()
def _set_molweight(self, value): pass
totalmolweight = property(_get_molweight,_set_molweight)

class Meta:
    db_table = u'molecularspecies'
def __unicode__(self):
    return_string = u""
    if self.name:
        return_string += u"name:␣" + self.name + u";␣"
    if self.formula:
        return_string += u"formula:␣" + self.formula + u";␣"
    if self.charge:
        return_string += u"charge:␣" + str(self.charge) + u";
␣"
    if len(return_string)>0:
        return_string = return_string[:-2]
    else:
        return_string = u"Inchi:␣" + self.inchi
    return return_string
def atoms(self):
    result = []
    for element in self.elementspecies_set.all():
        result.append(element)
    return result
def electrons_number(self):
    electrons = 0
    for tmp_atom in self.atoms():
        electrons += tmp_atom.electron_number()
    electrons += self.charge
    return electrons
def inchi_without_charge(self):
    if self.inchi:
        return self.inchi.split("/q")[0].strip()
    else:
        #Error
```

```
            return "Inchi undefined"
        def now(self, force=False):
            if force or not self.time_stamp:
                self.time_stamp = cf.datetimenow()
        def expire(self):
            return cf.dataexpire(self)
        def expired(self, timecheck = None):
            return cf.checkexpired(self, timecheck)
        def molweight(self):
            weight = 0
            if self.species_id:
                elements_species = ElementSpecies.objects.filter(
        species = self)
                for e in elements_species:
                    weight += e.element.standard_atomic_weight
            return weight
        def OrdinaryStructuralFormula(self):
            result = u""
            if self.formula:
                s = False
                for c in self.formula:
                    if re.match(r"[0-9]", c):
                        if not s:
                            result += u"\u005F{" + c
                            s = True
                        else:
                            result += c
                    else:
                        if s:
                            result += u"}" + c
                            s = False
                        else:
                            result += c
                if s:
                    result += u"}"
                result = u"$" + result  + u"$"
            else:
                result += u"$NOFORMULA$"
            return result

        expired.boolean = True
        objects = models.Manager() # The default manager.
        alive_objects = GenericModelNotExpiredManager()


    class ElementSpecies(models.Model):
        elemspecies_id = models.AutoField(primary_key=True)
        element = models.ForeignKey('Elements')
        species = models.ForeignKey('MolecularSpecies')
        number = models.IntegerField(null=True, blank=True)
```

```
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'elementspecies'


class Polarisabilities(models.Model):
    pol_id = models.AutoField(primary_key=True)
    state = models.ForeignKey('ElectronicStates')
    task = models.ForeignKey('Tasks')
    bibliographies = models.ManyToManyField('Bibliography')
    low_freq_lim = models.FloatField(null=True, blank=True)
    up_freq_lim = models.FloatField(null=True, blank=True)
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'polarisabilities'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class PolarisabilitiesBibliographies(models.Model):
    id = models.AutoField(primary_key= True)
    polarisabilities = models.ForeignKey('Polarisabilities')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'polarisabilities_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
```

```
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class PublicationSeries(models.Model):
    series_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length = 200, blank=True)
    shortname = models.TextField(max_length = 200, blank=True)
    comments = models.TextField(blank=True)
    class Meta:
        db_table = u'publicationseries'



class Publishers(models.Model):
    publisher_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length = 200, blank=True)
    address = models.TextField(blank=True)
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'publishers'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class Reftype(models.Model):
    type_id = models.AutoField(primary_key=True)
    description = models.CharField(max_length = 200,blank=True)
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'reftype'
    def __unicode__(self):
        if self.description:
            return self.description
```

```
            else:
                return u"NO␣DESCRIPTION"
        def now(self, force=False):
            if force or not self.time_stamp:
                self.time_stamp = cf.datetimenow()
        def expire(self):
            return cf.dataexpire(self)
        def expired(self, timecheck = None):
            return cf.checkexpired(self, timecheck)
        expired.boolean = True
        objects = models.Manager() # The default manager.
        alive_objects = GenericModelNotExpiredManager()


    class RotationalConstantsBibliographies(models.Model):
        id = models.AutoField(primary_key= True)
        rotationalcostants = models.ForeignKey('RotationalConstants')
        bibliography = models.ForeignKey('Bibliography')
        time_stamp = models.DateTimeField(null=True, blank=True)
        qual_index = models.IntegerField(null=True, blank=True,
        verbose_name = "Quality␣Index")
        comments = models.TextField(blank=True)
        time_exp = models.DateTimeField(null=True, blank=True)
        class Meta:
            db_table = u'rotationalconstants_bibliographies'
        def now(self, force=False):
            if force or not self.time_stamp:
                self.time_stamp = cf.datetimenow()
        def expire(self):
            return cf.dataexpire(self)
        def expired(self, timecheck = None):
            return cf.checkexpired(self, timecheck)
        expired.boolean = True
        objects = models.Manager() # The default manager.
        alive_objects = GenericModelNotExpiredManager()


    class TabulatedChis(models.Model):
        chiid = models.AutoField(primary_key=True)
        vibanalisysanarmonic = models.ForeignKey('
        VibrationalAnalysesAnarmonic')
        vib_id1 = models.ForeignKey('TabulatedVibrations', db_column=
        'vib_id1', related_name='vib_id6')
        vib_id2 = models.ForeignKey('TabulatedVibrations', db_column=
        'vib_id2', related_name='vib_id7')
        chi = models.FloatField(null=True, blank=True)
        class Meta:
            db_table = u'tabulatedchis'
```

```
class TabulatedPolarisabilities(models.Model):
    poltable_id = models.AutoField(primary_key=True)
    pol = models.ForeignKey('Polarisabilities')
    frequency = models.FloatField()
    re_alpha_xx = models.FloatField(null=True, blank=True)
    im_alpha_xx = models.FloatField(null=True, blank=True)
    re_alpha_yy = models.FloatField(null=True, blank=True)
    im_alpha_yy = models.FloatField(null=True, blank=True)
    re_alpha_zz = models.FloatField(null=True, blank=True)
    im_alpha_zz = models.FloatField(null=True, blank=True)
    re_alpha_xy = models.FloatField(null=True, blank=True)
    im_alpha_xy = models.FloatField(null=True, blank=True)
    re_alpha_xz = models.FloatField(null=True, blank=True)
    im_alpha_xz = models.FloatField(null=True, blank=True)
    re_alpha_yz = models.FloatField(null=True, blank=True)
    im_alpha_yz = models.FloatField(null=True, blank=True)
    class Meta:
        db_table = u'tabulated_polarisabilities'


class TheoryLevels(models.Model):
    thlevel_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length = 45, blank=True)
    description = models.TextField(blank=True)
    bibliographies = models.ManyToManyField('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True)
    xc_name = models.TextField(blank=True)
    xc_description = models.TextField(blank=True)
    comments = models.TextField(blank=True, null=True)
    class Meta:
        db_table = u'theorylevels'
    def __unicode__(self):
        returnstring = ""
        if self.name:
            returnstring += self.name
        else:
            returnstring += "NO␣NAME"
        if self.description:
            returnstring += "␣-␣" + self.description
        return returnstring

    #def fullprint(self):
    #    print "th_level_id:" + str(self.th_level_id) + "name:" +
     str(self.name) + "description:" + str(self.description) + "
    time_stamp:" + str(self.time_stamp) + "qual_index:"  + str(
    self.qual_index) + "comments:" + str(self.comments)


class Tasks(models.Model):
```

```
task_id = models.AutoField(primary_key=True)
thlevel = models.ForeignKey('TheoryLevels')
calc = models.ForeignKey('Calculations')
name = models.CharField(max_length=45)
description = models.TextField(blank=True)
time_stamp = models.DateTimeField(null=True, blank=True)
qual_index = models.IntegerField(null=True, blank=True)
comments = models.TextField(blank=True)
class Meta:
    db_table = u'tasks'
def __unicode__(self):
    returnstring = ""
    if self.name:
        returnstring += self.name
    else:
        returnstring += "NO␣NAME"
    if self.description:
        returnstring += "␣-␣" + self.description
    return returnstring
def usedbasissets(self):
    bs_list_id = []
    for bs in self.elementspeciesbasisset_set.all():
        if bs.basisset in bs_list_id:
            pass
        else:
            bs_list_id.append(bs.basisset)
    return bs_list_id
def returnmethoddescriptionandbib(self):
    methoddescription = ""
    bibliographies = []
    #Chemistry code
    methoddescription += "implementation:␣" + self.calc.code.
name + "␣" + self.calc.code.version + "\n"
    for bib in self.calc.code.bibliographies.all():
        if not (bib in bibliographies):
            bibliographies.append(bib)
    #Theory level
    methoddescription += "theory:␣" + self.thlevel.
description + "\n"
    if self.thlevel.xc_name:
        methoddescription += "xc:␣" + self.thlevel.
xc_description + "\n"
    for bib in self.thlevel.bibliographies.all():
        if not (bib in bibliographies):
            bibliographies.append(bib)
    #Basis sets
    bslist = self.usedbasissets()
    if len(bslist) > 0:
        methoddescription += "basis␣sets:␣"
        for bs in bslist:
```

```
                methoddescription += bs.name + ",␣"
                for bib in bs.bibliographies.all():
                    if not (bib in bibliographies):
                        bibliographies.append(bib)
            methoddescription = methoddescription[:-2] + "\n"
        return methoddescription, bibliographies



class ElectronicStates(models.Model):
    state_id = models.AutoField(primary_key=True)
    species = models.ForeignKey('MolecularSpecies')
    geom = models.ForeignKey('Geometries')
    task = models.ForeignKey('Tasks')
    bibliographies = models.ManyToManyField('Bibliography')
    total_energy = models.FloatField(null=True, blank=True)
    is_minimum = models.BooleanField(null=False)
    rel_energy = models.FloatField(null=True, blank=True)
    electronicstateenergy = models.ForeignKey("self", null=True,
    blank=True)
    symmetry = models.TextField(blank=True)
    multiplicity = models.IntegerField(null=True, blank=True)
    description = models.TextField(blank=True)
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True)
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'electronicstates'
    def __unicode__(self):
        returnstring = ""
        if self.total_energy:
            returnstring += "Total␣Energy:␣" + str(self.
    total_energy) + ";␣"
        if self.multiplicity:
            returnstring += "Multiplicity:␣" + str(self.
    multiplicity)
        return returnstring
    def countatoms(self):
        return self.species.atoms.count()
    def return_basissets_for_atoms(self):
        #return a list of tuple (atom, basisset)
        result = []
        esset = self.task.elementspeciesbasisset_set.all()
        for es_bs in esset.order_by('elementspecies__element').
    order_by('basisset'):
            result.append(es_bs)
        return result
    def equal_basis_set(self, basissets_for_atoms):
        mybasissets_for_atoms = self.return_basissets_for_atoms()
```

```
        result = True
        if len(mybasissets_for_atoms) == len(basissets_for_atoms)
    :
            for i in range(len(mybasissets_for_atoms)):
                if mybasissets_for_atoms[i].elementspecies.
    element != basissets_for_atoms[i].elementspecies.element or
    mybasissets_for_atoms[i].basisset != basissets_for_atoms[i].
    basisset:
                    result = False
                    break
        else:
            return False
        return result
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()



class RotationalConstants(models.Model):
    rot_id = models.AutoField(primary_key=True)
    state = models.ForeignKey('ElectronicStates')
    a = models.FloatField(null=True, blank=True)
    b = models.FloatField(null=True, blank=True)
    c = models.FloatField(null=True, blank=True)
    wilson_dj = models.FloatField(null=True, blank=True)
    wilson_djk = models.FloatField(null=True, blank=True)
    wilson_dk = models.FloatField(null=True, blank=True)
    nielsen_dj = models.FloatField(null=True, blank=True)
    nielsen_djk = models.FloatField(null=True, blank=True)
    nielsen_dk = models.FloatField(null=True, blank=True)
    nielsen_d_j = models.FloatField(null=True, blank=True)
    nielsen_r5 = models.FloatField(null=True, blank=True)
    nielsen_r6 = models.FloatField(null=True, blank=True)
    bibliographies = models.ManyToManyField('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'rotationalconstants'
    def now(self, force=False):
```

```
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class TheoryLevelsBibliographies(models.Model):
    id = models.AutoField(primary_key=True)
    theorylevels = models.ForeignKey('TheoryLevels')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'theorylevels_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class VanDerWaals(models.Model):
    vanderwaals_id = models.AutoField(primary_key=True)
    state_id_1 = models.ForeignKey('ElectronicStates', db_column=
    'state_id_1', related_name='state_id_1')
    state_id_2 = models.ForeignKey('ElectronicStates', db_column=
    'state_id_2', related_name='state_id_2')
    task = models.ForeignKey('Tasks')
    bibliographies = models.ManyToManyField('Bibliography')
    effective_freq = models.FloatField(null=True, blank=True)
    c6 = models.FloatField(null=True, blank=True)
    k = models.FloatField(null=True, blank=True)
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
```

```
    class Meta:
        db_table = u'vanderwaals'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class VanDerWallsBibliographies(models.Model):
    id = models.AutoField(primary_key= True)
    vanderwalls = models.ForeignKey('VanDerWaals')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'vanderwalls_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class VibrationalAnalysesAnarmonic(models.Model):
    vibanalysesanarmonic_id = models.AutoField(primary_key=True)
    polymode = models.TextField(blank=True)
    state = models.ForeignKey('ElectronicStates')
    task = models.ForeignKey('Tasks')
    bibliographies = models.ManyToManyField('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'vibrationalanalysesanarmonic'
```

```
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class Vibrationalanalysesharmonic(models.Model):
    vibrationalanalysesharmonic_id = models.AutoField(primary_key
    =True)
    state = models.ForeignKey('Electronicstates')
    task = models.ForeignKey('Tasks')
    bibliographies = models.ManyToManyField('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'vibrationalanalysesharmonic'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class TabulatedVibrations(models.Model):
    vib_id = models.AutoField(primary_key=True) #Probably problem
     whit bigint
    vibrationalanalysesharmonic = models.ForeignKey('
    Vibrationalanalysesharmonic')
    sym_type = models.TextField(blank=True)
    frequency = models.FloatField(null=True, blank=True)
    ir_intensity = models.FloatField(null=True, blank=True)
    alpha_a = models.FloatField(null=True, blank=True)
    alpha_b = models.FloatField(null=True, blank=True)
    alpha_c = models.FloatField(null=True, blank=True)
    diff_mu_x = models.FloatField(null=True, blank=True)
    diff_mu_y = models.FloatField(null=True, blank=True)
```

```
    diff_mu_z = models.FloatField(null=True, blank=True)
    eigenvectors = models.TextField(blank=True)
    class Meta:
        db_table = u'tabulatedvibrations'
    def __unicode__(self):
        return_string = ""
        if self.sym_type:
            return_string += "sym_type:␣" + str(self.sym_type) +
    ";␣"
        if self.frequency:
            return_string += "frequency:␣" + str(self.frequency)
    + ";␣"
        if self.alpha_a:
            return_string += "alpha_a:␣" + str(self.alpha_a) + ";
    ␣"
            return_string += "alpha_b:␣" + str(self.alpha_b) + ";
    ␣"
            return_string += "alpha_c:␣" + str(self.alpha_c) + ";
    ␣"
        if self.diff_mu_x:
            return_string += "diff_mu_x:␣" + str(self.diff_mu_x)
    + ";␣"
            return_string += "diff_mu_y:␣" + str(self.diff_mu_y)
    + ";␣"
            return_string += "diff_mu_z:␣" + str(self.diff_mu_z)
    + ";␣"
        if return_string:
            return return_string[:-2]
        else:
            return "Empty"


class VibrationalanalysesharmonicBibliographies(models.Model):
    id = models.AutoField(primary_key= True)
    vibrationalanalysesharmonic = models.ForeignKey('
    Vibrationalanalysesharmonic')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'vibrationalanalysesharmonic_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
```

```
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class VibrationalAnalysesAnarmonicBibliographies(models.Model):
    id = models.AutoField(primary_key= True)
    vibrationalanalysisanarmonic = models.ForeignKey('
    VibrationalAnalysesAnarmonic')
    bibliography = models.ForeignKey('Bibliography')
    time_stamp = models.DateTimeField(null=True, blank=True)
    qual_index = models.IntegerField(null=True, blank=True,
    verbose_name = "Quality␣Index")
    comments = models.TextField(blank=True)
    time_exp = models.DateTimeField(null=True, blank=True)
    class Meta:
        db_table = u'vibrationalanalysesanarmonic_bibliographies'
    def now(self, force=False):
        if force or not self.time_stamp:
            self.time_stamp = cf.datetimenow()
    def expire(self):
        return cf.dataexpire(self)
    def expired(self, timecheck = None):
        return cf.checkexpired(self, timecheck)
    expired.boolean = True
    objects = models.Manager() # The default manager.
    alive_objects = GenericModelNotExpiredManager()


class ElementSpeciesBasisSet(models.Model):
    elementspecies_basisset_id = models.AutoField(primary_key=
    True)
    basisset = models.ForeignKey('BasisSets')
    task = models.ForeignKey('Tasks')
    elementspecies = models.ForeignKey('ElementSpecies')

    class Meta:
        db_table = u'elementspecies_basisset'
```

Riferimenti bibliografici

[1] https://www.djangoproject.com/

[2] http://www.nwchem-sw.org/index.php/Main_Page

[3] http://www.tddft.org/programs/octopus/wiki/index.php/Main_Page

[4] http://www.gaussian.com/

[5] http://www.vamdc.eu/access-data/portal/

[6] http://it.wikipedia.org/wiki/Metodo_MCSCF

[7] http://it.wikipedia.org/wiki/Metodo_Coupled_Cluster

[8] http://it.wikipedia.org/wiki/Teoria_del_funzionale_della_densit\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{a\global\mathchardef\accent@spacefactor\spacefactor}\accent18a\egroup\spacefactor\accent@spacefactor

[9] http://it.wikipedia.org/wiki/Quantum_Monte_Carlo

[10] https://www.mysql.it/

[11] http://www.awstats.org/

[12] "PAH databases in VAMDC (Virtual Atomic and Molecular Data Center)", A.Saba, T.Louge, H.Sabbah, G.Mulas, C.Joblin - VAMDC Final Project Meeting - Poster - 14-16 Nov 2012 Meudon, Francia

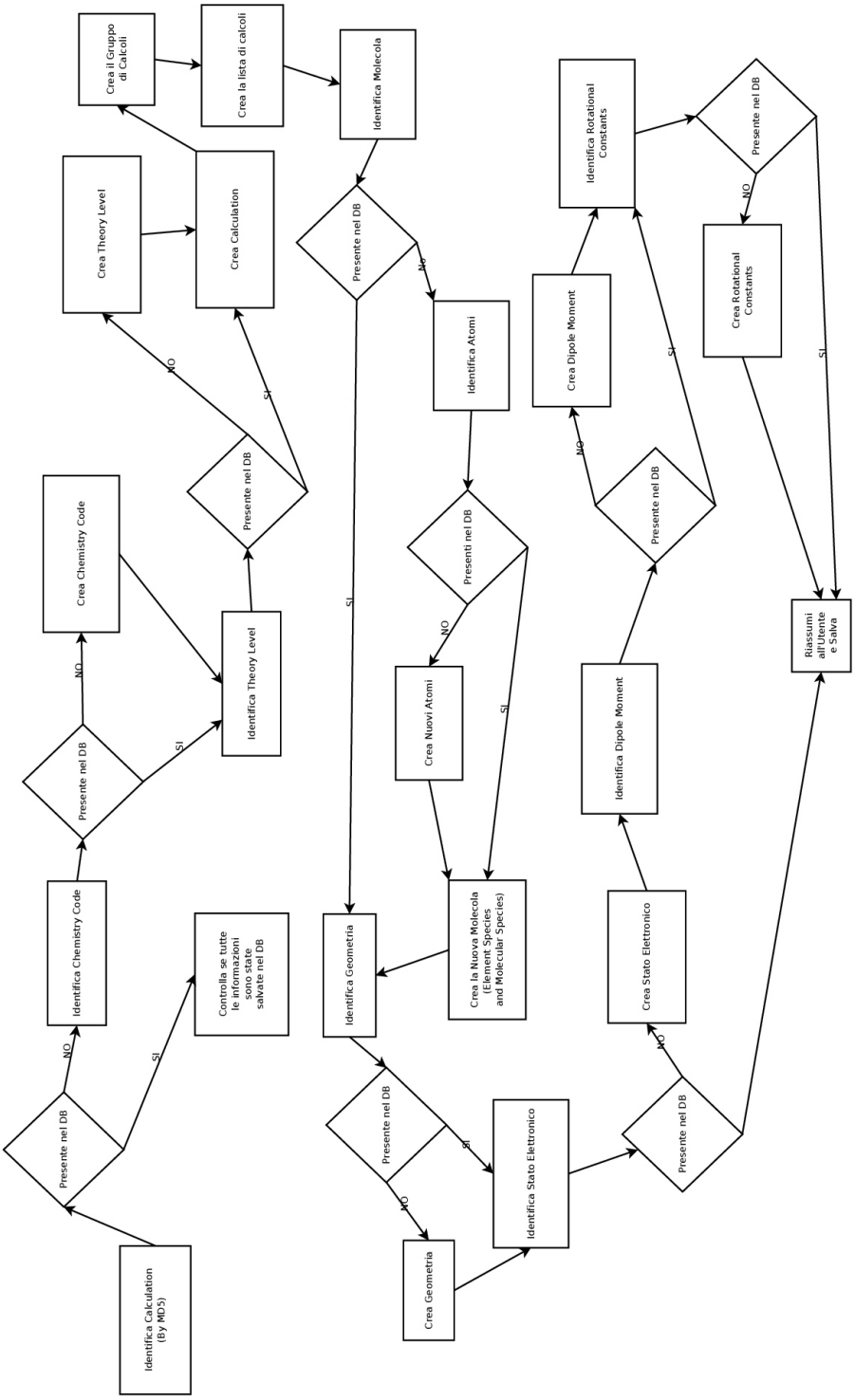

Figura 2: QChiTool Flow chart

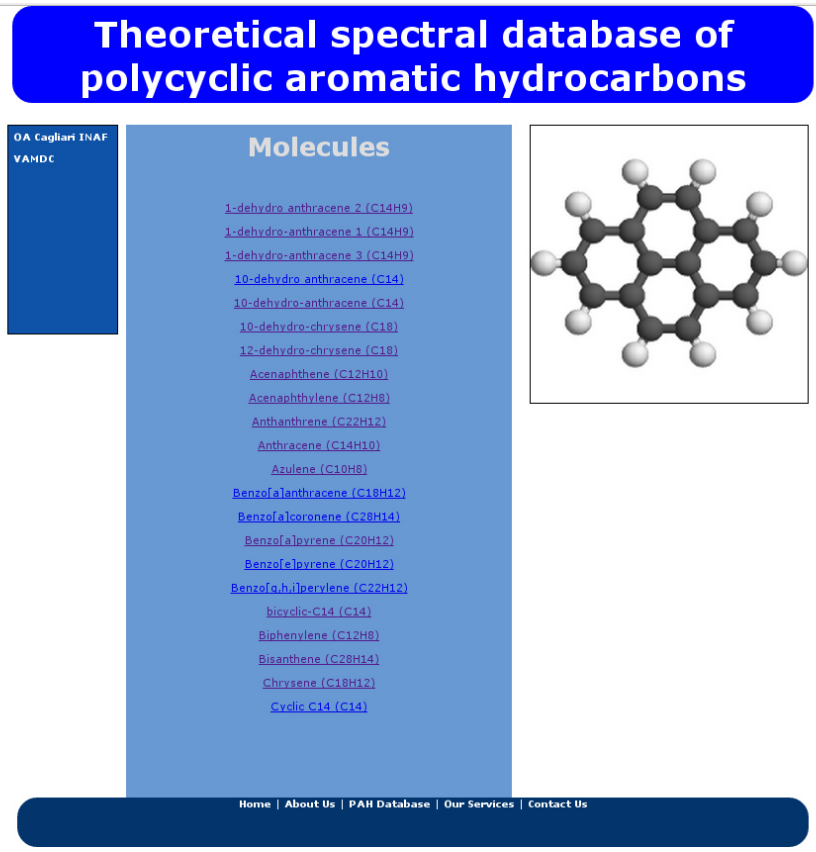

Figura 3: Frontend homepage

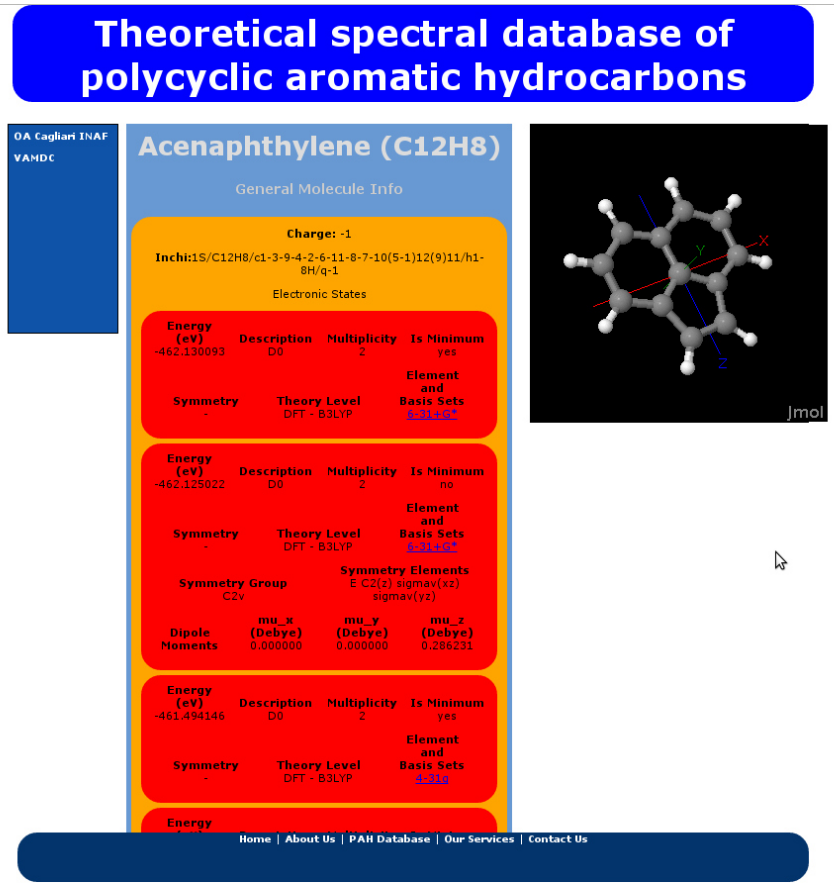

Figura 4: Electronic states info

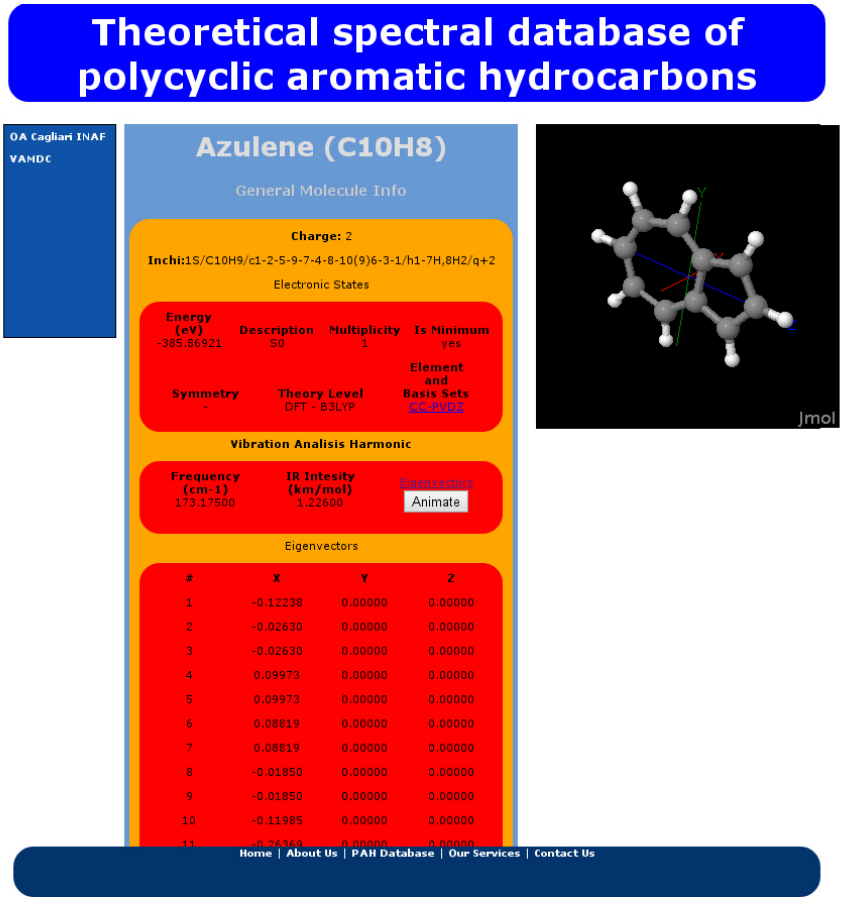

Figura 5: Vibrational analises info

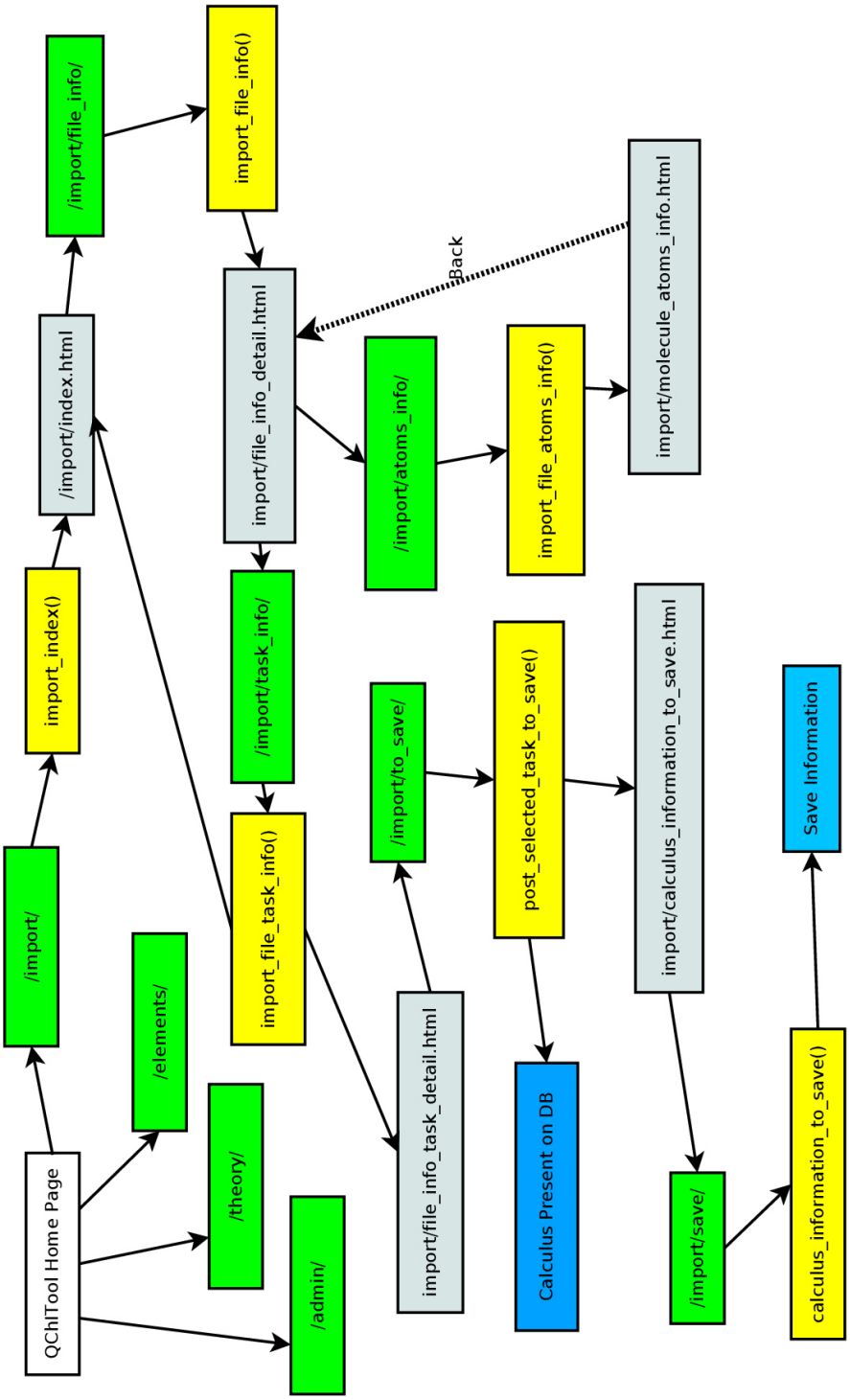

Figura 6: User flow chart

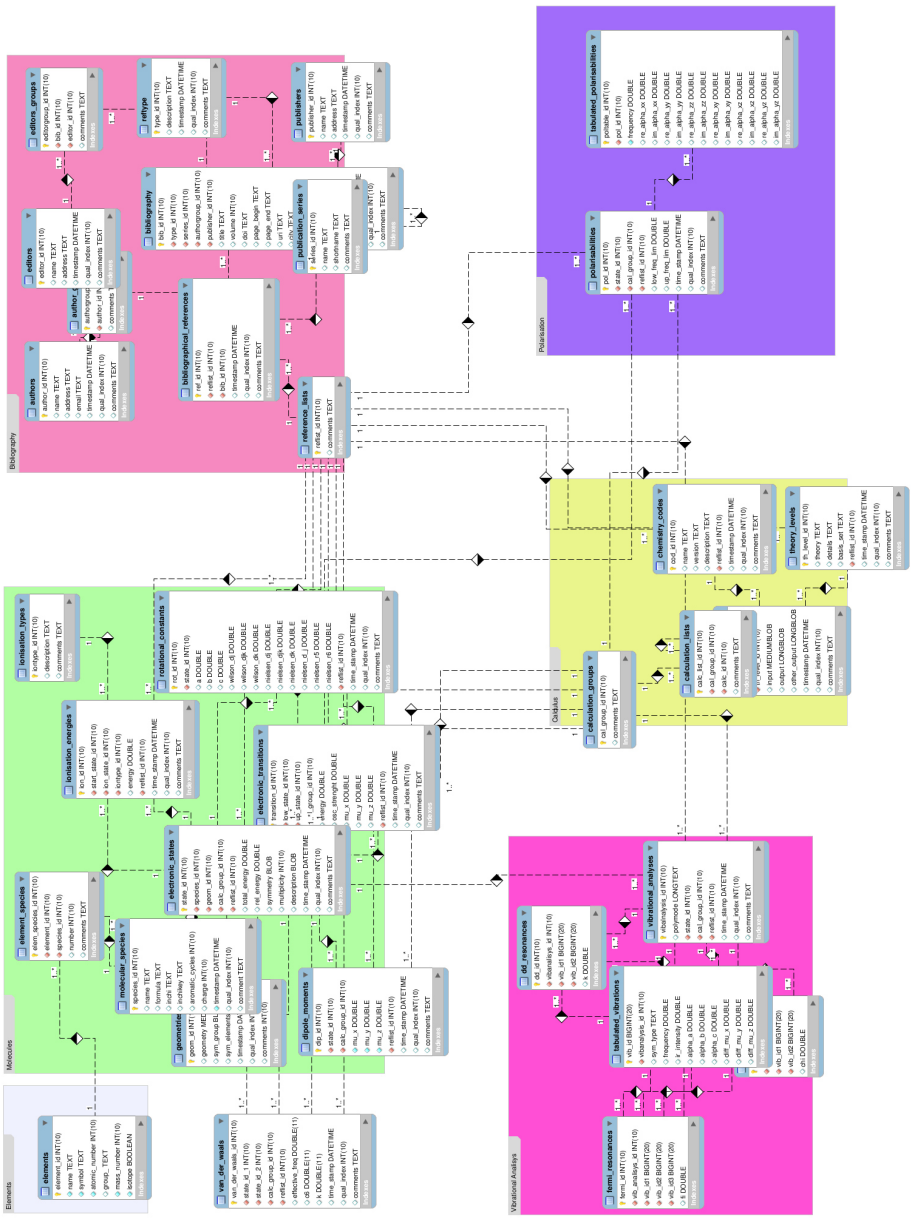

Figura 7: Complete database schema

Design of proposed PAH database structure

A. Saba, G. Mulas, G. Malloci

version 0.9.1, January 2011

Introduction

After the implementation of the first test implementation of atomic and molecular databases in a tentative VAMDC structure, it has been by and large decided that all database providers must (within the second year of the project) implement a RDBMS and insert their data into it. The common understanding, for several reasons, is that unless there are specific reasons warranting a different decision, MySQL will be the recommended underlying database engine.

The PAH dataset includes a number of heterogeneous information about different electronic states of different molecules. As to the adopted VAMDC baseline requirements, in general all pieces of information must preserve reproducibility at a given time and appropriate bibliographical references. Reproducibility implies that a datum must never be simply replaced by a better one, but instead both the old one and the new one must coexist, with a well defined timestamp and one or more quality indexes. In this way, a common query will return just the "best" datum (as defined by the appropriate quality index for the intended use). On the other hand, if one wants to know what would the answer of the database have been at a given moment in time, one can trivially obtain that by restricting the query by a constraint on timestamps. The implication is that each and every datum may well have several instances, and therefore a "simple" single (MySQL) table structure in which both molecular species and (some) quantities are listed is unfeasible. Indeed, there will necessarily be a large number of tables, connected by relations.

The structure of tables and their relations

The molecular_species table

This will be the "root" table, it will list unique species, as defined by chemical structure and charge. Additional fields like e. g. S_atoms might be added afterwards, should the need arise.

Field	**Type**	**Description**	**Properties**
species_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
name	TEXT	Name of the molecule	
formula	TEXT	Brute formula	Not null
inchi	TEXT	InChI code	
inchikey	TEXT	InChI key	
aromatic_cycles	INTEGER	Number of aromatic cycles	Unsigned, not null
charge	INTEGER	Charge in electrons	Not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The molecular_isotopologues table

This will be the list of isotopologues (if any) of a given molecular species

Field	Type	Description	Properties
isotopologues_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
species_id	TEXT	Corresponding to a molecular species	
inchi	BLOB	Specific InChI code of the isotopologue	
inchikey	BLOB	Specific InChI key of the isotopologue	
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The elements table

This table will list chemical elements.

Field	Type	Description	Properties
element_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
name	TEXT	Name of the element	Not null
symbol	TEXT	Element symbol	Not null
atomic_number	INTEGER	Atomic number	Unsigned, not null
group	TEXT	Group of element	Not null

The isotopes table

This table will list chemical elements.

Field	Type	Description	Properties
isotope_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
element_id	INTEGER	Corresponding to an element in the elements table	Not null
mass_number	INTEGER	Atomic mass number	Unsigned, not null
comments	TEXT	General comments (stability etc.)	

The element_species table

The element_species table will list for each molecular species (species_id) then number of chemical element (element_id) present.

Field	Type	Description	Properties
elem_species_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
element_id	INTEGER	Corresponding to a element_id in elements table	Unsigned, not null
species_id	INTEGER	Corresponding to a species_id in molecular_species table	Unsigned, not null
Number	INTEGER	Number of element in specie	Unsigned, not null
comments	TEXT	General comments	

The element_isotopologues table

The element_species table will list for each molecular species (species_id) then number of specific isotopes of chemical elements (isotope_id) present.

Field	Type	Description	Properties
element_isotopologues_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
isotope_id	INTEGER	Corresponding to a isotope_id in isotopes table	Unsigned, not null
isotopologues_id	INTEGER	Corresponding to a species_id in molecular_isotopologues table	Unsigned, not null
number	INTEGER	Number of isotope in specie	Unsigned, not null
comments	TEXT	General comments	

The electronic_states table

This table will list electronic states of a given molecule. For many molecules, only the ground electronic state will be present.

Field	Type	Description	Properties
state_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
species_id	INTEGER	Corresponding to a species_id in molecular_species table	Unsigned, not null
geom_id	INTEGER	Corresponding to a geom_id in geometries table	Unsigned, not null
calc_group_id	INTEGER	Corresponding to a calc_group_id in calculation_groups table	Unsigned
reflist_id	INTEGER	Corresponding to a reflist_id in bibliographical_references table	Unsigned, not null
total_energy	DOUBLE	Total energy of the electronic state	Not null
rel_energy	DOUBLE	Energy relative to the ground electronic state	
symmetry	BLOB	Symmetry of the electronic state	Not null
multiplicity	INTEGER	Spin multiplicity of the electronic state	Not null
description	BLOB	Description of the electronic state, e. g. in terms of electronic molecular orbitals.	
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The geometries table

This table lists molecular geometries, in the (ask Giuliano) unambiguous format. This format is human-readable text and can be easily converted from/to other common formats by the OpenBabel open source toolset.

Field	Type	Description	Properties
geom_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
geometry	MEDIUMBLOB	Molecular geometry in SDF format	Not null
sym_group	BLOB	Symmetry group of this molecular configuration	Not null
sym_elements	BLOB	Symmetry elements, referred to this configuration	Not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The calculations, calculation_lists and calculation_groups tables

The calculations table lists specific runs of quantum chemistry programs, giving some basic information (level of theory and specific code used) and, whenever available, complete input and output files. The input files can be useful for reproducibility, the output files can be useful for visualisation, since some codes (e. g. Jmol) can directly read them and display the results they contain (e. g. vibrational modes). The calculation_lists table is meant to be used to link together (and thus be able to referer to) groups of strictly related calculations, such as e. g. the three octopus runs to obtain all components of the dynamic polarisability tensor. The calculation_groups table lists these groups.

Field	Type	Description	Properties
calc_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
code_id	INTEGER	Code used, matches a code_id in chemistry_codes table	Unsigned, not null
th_level_id	INTEGER	Theory level used, matches a th_level_id in theory_levels table	Unsigned, not null
input	MEDIUMBLOB	Input given to the code	
output	LONGBLOB	Main output produced by the code	
other_output	LONGBLOB	Other output produced by the code	
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

Field	Type	Description	Properties
calc_list_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
calc_group_id	INTEGER	Relates a group of strictly related calculations	Unsigned, not null
calc_id	INTEGER	Matches a calc_id entry in calculations table	Unsigned, not null
comments	TEXT	General comments	

Field	Type	Description	Properties
calc_group_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
comments	TEXT	General comments	

The chemistry_codes table

This table lists the quantum chemistry codes used for calculations in the database. Enough information is included to enable one to find the exact code used for each calculation in the database.

Field	Type	Description	Properties
code_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
name	TEXT	Name of the quantum chemistry code	Not null
version	TEXT	Version string	
description	TEXT	Short description of the code, possibly including an official URL for it	Not null
reflist_id	INTEGER	Relates a list of references for this quantum chemistry code	Unsigned, index, not null
timestamp	DATETIME	Time of insertion	Unsigned, not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The theory_levels table

This table lists “levels of theory”, with enough detail to enable one to reproduce a given calculation.

Field	Type	Description	Properties
th_level_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
theory	TEXT	Short name of the theory used (e. g. DFT)	Not null
details	TEXT	Theory details, such as exchange-correlation functional in DFT, use of approximations as resolution of identity etc.	
basis_set	TEXT	Basis set used (e. g. 4-31G for a Gaussian basis set, or a grid shape/size/step)	Not null
reflist_id	INTEGER	Corresponding to a reflist_id in bibliographical_references table	Unsigned, index
timestamp	DATETIME	Time of insertion	Unsigned, not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The authors and author_groups table

This table lists authors for all references, and groups of them participating to a given publication

Field	Type	Description	Properties
author_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
name	TEXT	Name of the author	Not null
address	TEXT	Institute and address of the author	Not null
email	TEXT	Email address	Not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

Field	Type	Description	Properties
authorgroup_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
bib_id	INTEGER	Corresponding publication in bibliography table	Not null
author_id	INTEGER	Corresponding author in authors table	Not null
comments	TEXT	General comments	

The reftype table

This table lists the types of references (sources) for the data, e. g. journal, book, personal communication, URL, etc.

Field	Type	Description	Properties
type_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
description	TEXT	Description of this type of reference	Not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The editors and editor_groups table

This table lists the possible editors of the publications and how they are grouped in editing specific publications

Field	Type	Description	Properties
editor_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
name	TEXT	Name of the editor	Not null
address	TEXT	Address of the editor	Not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

Field	Type	Description	Properties
editorgroup_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
bib_id	INTEGER	Corresponding publication in bibliography table	Not null
editor_id	INTEGER	Corresponding editor in editors table	Not null
comments	TEXT	General comments	

The publishers table

This table lists the publishers of the refereces

Field	Type	Description	Properties
publisher_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
name	TEXT	Publisher name	Not null
address	TEXT	Address of the publisher	Not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The publication_series table

This table lists the publication series, such as journals, so that any given journal is always called with the same name

Field	Type	Description	Properties
series_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
name	TEXT	Full name of series (e. g. Monthly Notices of the Royal Astronomical Society)	
shortname	TEXT	Short version to be used in bibliographical references (e. g MNRAS)	
comments	TEXT	General comments	

The bibliography table

This table lists all bibliographical references used in the database.

Field	Type	Description	Properties
bib_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
type_id	INTEGER	Corresponding to a type of reference in reftype table	
series_id	INTEGER	Corresponding to one record in the table listing regular publication series (e. g. journals)	
title	TEXT	Title of the publication	
volume	INTEGER	Volume in the series	
doi	TEXT	DOI of the publication	
page_begin	TEXT	Beginning page	
page_end	TEXT	End page	
uri	TEXT	URI of the publication	
city	TEXT	City of publication	
version	TEXT	Version of publication	
source_name	TEXT	Bibliographic reference name (e.g. journal)	
date	DATE	Date of publication	Not null
reference	TEXT	Complete reference in free text format	Not null
bibtex	TEXT	Complete reference in BibTeX format	Not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The bibliographical_references table

This table lists just two keys to establish a relation by a datum and the list of references to be cited for it.

Field	Type	Description	Properties
ref_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
reflist_id	INTEGER	Relates a list of references for a given datum	Unsigned, index, not null
bib_id	INTEGER	Corresponds to a bib_id in bibliography table	Not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The reference_lists table

This table lists the groups of references

Field	Type	Description	Properties
reflist_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
comments	TEXT	General comments	

The ionisation_energies table

This table lists the ionisation energies of a given electronic state of a given molecule

Field	Type	Description	Properties
ion_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
start_state_id	INTEGER	Corresponding to a state_id in the molecular_states table	Unsigned, index, not null
ion_state_id	INTEGER	Corresponding to a state_id in the molecular_states table	Unsigned, index
iontype_id	INTEGER	Corresponding to a iontype	Not null
energy	DOUBLE	The ionisation energy value (in eV)	Not null
reflist_id	INTEGER	Corresponding to a reflist_id in bibliographical_references table	Unsigned, not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The ionisation_types table

This table lists the acceptable values for the “type” field in the ionisation_energies table

Field	Type	Description	Properties
iontype_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
description	TEXT	Description of the ionisation type (e. g. single experimental ionisation energy)	Not null
comments	TEXT	General comments	

The rotational_constants table

This table lists the rotational constants (rigid rotor approximation) in a given electronic state

Field	Type	Description	Properties
rot_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
state_id	INTEGER	Corresponding to a state_id in the molecular_states table	Unsigned, index, not null
isotopologues_id	INTEGER	Corresponding to a isotopologues_id in the molecular_isotopologues table	Unsigned, index, not null
A	DOUBLE	A rotational constant (in cm^{-1})	Unsigned, not null
B	DOUBLE	B rotational constant (in cm^{-1})	Unsigned, not null
C	DOUBLE	C rotational constant (in cm^{-1})	Unsigned, not null
Wilson_DJ	DOUBLE	Wilson DJ quartic term	
Wilson_DJK	DOUBLE	Wilson DJK quartic term	
Wilson_DK	DOUBLE	Wilson DK quartic term	
Nielsen_DJ	DOUBLE	Nielsen_DJ quartic term	
Nielsen_DJK	DOUBLE	Nielsen_DJK quartic term	
Nielsen_DK	DOUBLE	Nielsen_DK quartic term	
Nielsen_dJ	DOUBLE	Nielsen_dJ quartic term	
Nielsen_R5	DOUBLE	Nielsen_R5 quartic term	
Nielsen_R6	DOUBLE	Nielsen_R6 quartic term	
reflist_id	INTEGER	Corresponding to a reflist_id in bibliographical_references table	Unsigned, index, not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The dipole_moments table

This table lists the permanent (static) electric dipole moments in a given electronic state

Field	Type	Description	Properties
dip_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
state_id	INTEGER	Corresponding to a state_id in the molecular_states table	Unsigned, index, not null
calc_group_id	INTEGER	Corresponding to a calc_group_id in calculation_groups table	Unsigned
mu_x	DOUBLE	μ_x rotational constant (in Debye)	Not null
mu_y	DOUBLE	μ_y rotational constant (in Debye)	Not null
mu_z	DOUBLE	μ_z rotational constant (in Debye)	Not null
reflist_id	INTEGER	Corresponding to a reflist_id in bibliographical_references table	Unsigned, not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The polarisabilities and tabulated_polarisabilities tables

The polarisability table lists the calculations of dynamic electronic polarisability for a given electronic state. Three calculations are linked to, since up to three Octopus runs are needed to obtain the full tensor. The actual, tabulated polarisability tensors as a function of frequency, are given in tabulated_polarisabilities table. The electronic (unpolarised) photo-absorption cross-section is proportional to the trace of the imaginary part of this tensor at the same frequency. A function is defined in MySQL to obtain a view of this table listing directly the photo-absorption cross-sections.

Field	Type	Description	Properties
pol_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
state_id	INTEGER	Corresponding to a state_id in the molecular_states table	Unsigned, index, not null
calc_group_id	INTEGER	Corresponding to a calc_group_id in calculation_lists table	Unsigned, not null
reflist_id	INTEGER	Corresponding to a reflist_id in bibliographical_references table	Unsigned, index, not null
low_freq_lim	DOUBLE	The lower limit of the frequency interval covered by this calculation	Unsigned, not null
up_freq_lim	DOUBLE	The upper limit of the frequency interval covered by this calculation	Unsigned, not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

Field	Type	Description	Properties
poltable_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
pol_id	INTEGER	Corresponding to a pol_id in the polarisabilities table	Unsigned, index, not null
frequency	DOUBLE	Frequency (in cm^{-1}, zero means static polarisability)	Unsigned, not null
re_alpha_xx	DOUBLE	Real part of α_{xx} polarisability (in $Å^3$)	Not null
im_alpha_xx	DOUBLE	Imaginary part of α_{xx} polarisability (in $Å^3$)	Not null
re_alpha_yy	DOUBLE	Real part of α_{yy} polarisability (in $Å^3$)	Not null
im_alpha_yy	DOUBLE	Imaginary part of α_{yy} polarisability (in $Å^3$)	Not null
re_alpha_zz	DOUBLE	Real part of α_{zz} polarisability (in $Å^3$)	Not null
im_alpha_zz	DOUBLE	Imaginary part of α_{zz} polarisability (in $Å^3$)	Not null
re_alpha_xy	DOUBLE	Real part of α_{xy} polarisability (in $Å^3$)	Not null
im_alpha_xy	DOUBLE	Imaginary part of α_{xy} polarisability (in $Å^3$)	Not null
re_alpha_xz	DOUBLE	Real part of α_{xz} polarisability (in $Å^3$)	Not null
im_alpha_xz	DOUBLE	Imaginary part of α_{xz} polarisability (in $Å^3$)	Not null
re_alpha_yz	DOUBLE	Real part of α_{yz} polarisability (in $Å^3$)	Not null
im_alpha_yz	DOUBLE	Imaginary part of α_{yz} polarisability (in $Å^3$)	Not null

The van_der_waals table

The van_der_waals table lists orientation-averaged van der Waals coefficients for the long-range interaction between pairs of molecules (in a given electronic state, usually the ground one). They are obtained from the frequency-dependent complex polarisabilities, from which orientation-specific coefficients can also be obtained, in principle, by the user.

Field	Type	Description	Properties
poltable_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
state_id_1	INTEGER	Corresponding to a state_id in the molecular_states table	Unsigned, index, not null
state_id_2	INTEGER	Corresponding to a state_id in the molecular_states table	Unsigned, index, not null
calc_group_id	INTEGER	Corresponding to a calc_group_id in calculation_lists table	Unsigned, not null
reflist_id	INTEGER	Corresponding to a reflist_id in bibliographical_references table	Unsigned, index, not null
effective_freq	DOUBLE	Effective frequency for the onset of retarded regime	
c6	DOUBLE	C_6 Hamamaker coefficient	
K	DOUBLE	Coefficient of retarded interaction	
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

The (harmonic) vibrational_analyses, tabulated_vibrations, tabulated_chis, fermi_resonances, dd_resonances tables.

The vibrational_analyses table lists the harmonic vibrational analyses for a given electronic state (of a given molecule). Actual frequencies and IR intensities are given in the related tabulated_vibrations table. Moreover, where available, the tabulated_chis table lists the anharmonic (deperturbed) χ_{ij} values for all normal modes; in this case, Fermi and Darling-Dennison resonances which were taken into account in computing deperturbed χ_{ij} are listed in the appropriate tables.

Field	Type	Description	Properties
vibanalysis_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
polymode	LONGTEXT	POLYMODE format values of second, third, fourth order force constants	
state_id	INTEGER	Corresponding to a state_id in the molecular_states table	Unsigned, index, not null
calc_group_id	INTEGER	Corresponding to a calc_group_id in calculation_groups table	Unsigned, not null
reflist_id	INTEGER	Corresponding to a reflist_id in bibliographical_references table	Unsigned, index, not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	

Field	Type	Description	Properties
vib_id	BIGINT	Unique index	Autoincrement, unsigned, primary index, not null
vibanalysis_id	INTEGER	Corresponding to a vibanalysis_id in the vibrational_analyses table	Unsigned, index, not null
sym_type	TEXT	Symmetry type of this vibrational mode	
frequency	DOUBLE	Harmonic frequency (in cm^{-1})	Unsigned, not null
ir_intensity	DOUBLE	Intensity of the IR transition in absorption (km mol-1)	Unsigned, not null
alpha_a	DOUBLE	Anharmonic/Coriolis correction to A rotational constant for this mode	
alpha_b	DOUBLE	Anharmonic/Coriolis correction to B rotational constant for this mode	
alpha_c	DOUBLE	Anharmonic/Coriolis correction to C rotational constant for this mode	
diff_mu_x	DOUBLE	x component of the derivative of the electric dipole moment along this normal vibrational mode (in Debye)	
diff_mu_y	DOUBLE	y component of the derivative of the electric dipole moment along this normal vibrational mode (in Debye)	
diff_mu_z	DOUBLE	z component of the derivative of the electric dipole moment along this normal vibrational mode (in Debye)	

Field	Type	Description	Properties
chi_id	BIGINT	Unique index	Autoincrement, unsigned, primary index, not null
vibanalysis_id	INTEGER	Corresponding to a vibanalysis_id in the vibrational_analyses table	Unsigned, index, not null
vib_id1	BIGINT	Corresponding to a vib_id in tabulated_vibrations (i index)	
vib_id2	BIGINT	Corresponding to a vib_id in tabulated_vibrations (j index)	
chi	DOUBLE	Value of χ_{ij} (cm-1)	

Field	Type	Description	Properties
fermi_id	BIGINT	Unique index	Autoincrement, unsigned, primary index, not null
vibanalysis_id	INTEGER	Corresponding to a vibanalysis_id in the vibrational_analyses table	Unsigned, index, not null
vib_id1	BIGINT	Corresponding to a vib_id in tabulated_vibrations (i index)	
vib_id2	BIGINT	Corresponding to a vib_id in tabulated_vibrations (j index)	
vib_id3	BIGINT	Corresponding to a vib_id in tabulated_vibrations (k index)	
FI	DOUBLE	Value of the FI(i,j,k) force constant (cm-1)	

Field	Type	Description	Properties
dd_id	BIGINT	Unique index	Autoincrement, unsigned, primary index, not null
vibanalysis_id	INTEGER	Corresponding to a vibanalysis_id in the vibrational_analyses table	Unsigned, index, not null
vib_id1	BIGINT	Corresponding to a vib_id in tabulated_vibrations (i index)	
vib_id2	BIGINT	Corresponding to a vib_id in tabulated_vibrations (j index)	
K	DOUBLE	Value of the K(i,i,j,j) force constant (cm-1)	

The electronic_transitions table

This table lists electronic transitions, as computed e, g. by NWChem or Turbomole. It includes the state_id of the lower and upper states involved, the energy of the transition (even if redundant?) the oscillator strength, possibly the transition electric dipole moment vector (referred to the geometry of the lower state).

Field	Type	Description	Properties
transition_id	INTEGER	Unique index	Autoincrement, unsigned, primary index, not null
low_state_id	INTEGER	Corresponding to a state_id in the molecular_states table	Unsigned, index, not null
up_state_id	INTEGER	Corresponding to a state_id in the molecular_states table	Unsigned, index, not null
calc_group_id	INTEGER	Corresponding to a calc_group_id in the calculation_groups table	Unsigned, not null
energy	DOUBLE	The transition energy (in eV)	Not null
osc_strength	DOUBLE	Oscillator strength	Unsigned, not null
mu_x	DOUBLE	x component of the the transition electric dipole moment (in Debye)	
mu_y	DOUBLE	y component of the the transition electric dipole moment (in Debye)	
mu_z	DOUBLE	z component of the the transition electric dipole moment (in Debye)	
reflist_id	INTEGER	Corresponding to a reflist_id in bibliographical_references table	Unsigned, not null
timestamp	DATETIME	Time of insertion	Not null
qual_index	INTEGER	Quality index	Unsigned, not null
comments	TEXT	General comments	