Documentazione Tecnica Laboratorio B Progetto Centri Vaccinali

**INDICE GENERALE**

Strumenti e linguaggi utilizzati............................................................................................................2

Processo di sviluppo...........................................................................................................................3

Suddivisione dei lavori........................................................................................................................4

Progettazione......................................................................................................................................4

Strumenti di sviluppo :

• JDK 18

• IDE per java: Eclipse, con plugin :

◦ Window builder(Java Swing) per lo sviluppo delle interfacce grafiche

• IDE per xml e html: blocco note(come si faceva ai miei tempi)

• GitHub Desktop per la gestione delle repository locali e online

• PostgreSQL 14.0 come dbms

• PGAdmin 4 per l’interazione con il database

• ? per la realizzazione dei diagrammi UML

• ? DraftSight e DIA per la realizzazione dei diagrammi ER

Linguaggi utilizzati:

• java per il core dell’applicazione, sia client che server

• sql per la realizzazione del database e per le query

• xml per la configurazione del POM di Maven

API:

• Postgresql 9.4 JDBC per l’interazione del lato server con il database

**PROCESSO DI SVILUPPO**

Dopo un’analisi delle specifiche di laboratorio ci siamo concentrati sulla discussione delle possibili scelte implementative riguardanti la realizzazione dell’architettura. Tra le varie soluzioni considerate è stata scelta la seguente ? struttura :

* **Lato client** → implementazione di un’interfaccia grafica che potesse offrire sia i servizi :
  + al cittadino : Ricerca centro vaccinale e visualizzazione delle sue informazioni, iscrizione per accedere all’area riservata dove segnalare eventuale eventi avversi
  + all’operatore sanitario : Creazione nuovo centro vaccinale, Registrazione di un neo vaccinato

**Lato server** → gestione multi-thread delle richieste da parte dei client e comunicazione con un DB relazionale per la gestione dei dati relativi a centri vaccinali, vaccinati e cittadini.

Su questa base sono stati poi creati i diagrammi UML ed ER.

Successivamente si è deciso di implementare una versione basilare del progetto che si concentrasse soprattutto sulla comunicazione server-db, per verificare l’effettiva funzionalità delle scelte implementative e per prendere familiarità con l’API JDBC. E’ stata quindi impostata una solida struttura della classe server.

Parallelamente è stata creata una versione basilare del client, senza interfaccia grafica, per testare l’efficacia e la comunicazione con la classe Server.

Avendo ottenuto buoni risultati è stato deciso di implementare in modo completo la struttura del client andando a creare e suddividere le classi appartenenti alle entità “cittadino” e “operatore sanitario”.

Si è poi passati all’implementazione di un efficiente metodo di comunicazione tra client e server.

Sono stati implementati lato client l’analisi e l’invio delle richieste, aggiungendo controlli sugli input utente.

Successivamente ci siamo concentrati sull’integrazione del progetto con Maven implementando le funzionalità richieste.

Abbiamo poi testato il progetto sia tramite IDE che tramite comandi Maven e non riscontrando problemi siamo passati alla realizzazione della documentazione.

**Suddivisione dei lavori**

MOI MATTEO : lato client e comunicazione client server

RABUFFETTI ALEX : lato server e comunicazione server db

Durante tutto il percorso di sviluppo c’è stata una costante comunicazione tra le parti per velocizzare il raggiungimento dell'obiettivo e una buona ottimizzazione ed efficienza del codice.

**CLIENT**

**package centrivaccinali**

**CentriVaccinali** : Questa è la classe main con la quale lo user si interfacci appena avviato il programma e dove seleziona il suo ruolo, ovvero se “operatore sanitario” o “cittadino”. E’ inoltre presente il metodo openSocket(), che permette la creazione di un nuovo socket. Tale metodo non viene richiamato da questa classe ma viene utilizzato da altre classi presenti nei package centrivaccinali e cittadini. OpenSocket() è fondamentale poichè, indipendentemente dal ruolo scelto dallo user, permette la corretta comunicazione tra server e client, andando a creare una nuova connessione per ogni nuova richiesta inviata dal client. Ciò rende indipendente sia le classi che le singole richieste.

**CentroVaccinale** : Questa classe va a definire un generico centro vaccinale e contiene un costruttore per l’oggetto CentroVaccinale, utilizzato dalla classe IscrizioneCentroVax. Sono presenti anche metodi get e set per ottenere specifici dati relativi ad un’istanza di CentroVaccinale.

Inoltre è presente il metodo getInfo() che, oltre a restituire i dati generici dell’istanza, mostra anche la severità media e il numero di segnalazioni di eventi avversi relativi al singolo centro vaccinale.

**ConnessioneServer** : Tale classe viene impiegata per la creazione di nuove connessioni tra client e server tramite il metodo richiesta Server(). Per la creazione di una nuova istanza è necessario specificare i parametri socket, una stringa contenente il tipo di richiesta e un oggetto generico che verrà poi identificato dal server grazie al tipo di richiesta inviata dal client.

E’ presente anche il metodo cerca Centro Vaccinale() , il quale permette la ricerca di un centro vaccinale registrato nel DB salvandone il risultato in un ArrayList di tipo Centro Vaccinale. E’ stata scelta questa struttura dati poichè, per come dovrebbe funzionare la ricerca, il numero risultante di centri vaccinali è basso e l’ArrayList permette una facile e dinamica consultazione di ogni centro.

ricezioneServer() serve per leggere la risposta del server in seguito alla elaborazione della richiesta del client, in particolare comunica l’id univoco dei neo-vaccinati e l’esito dell'operazione di login.

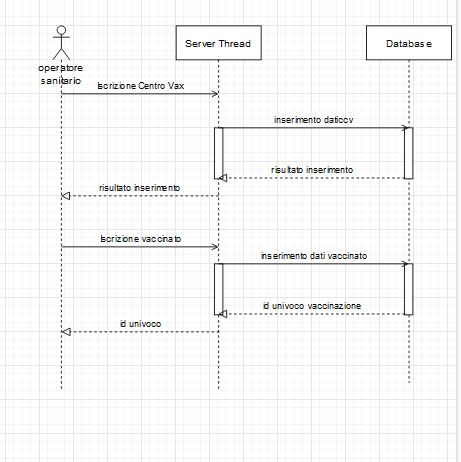
**IscrizioneCentroVax** : La classe prende in input i dati richiesti per la registrazione di un nuovo centro vaccinale nel DB e li invia al server che eseguirà tale operazione.

**IscrizioneVaccinato** : In questa classe è previsto che l’operatore cerchi il centro vaccinale dove il cittadino si sta vaccinando e dopo la selezione di quest’ultimo, verranno inseriti i restanti dati del vaccinato.

Tali dati verranno poi inviati al server. Dopo la registrazione del vaccinato nel DB, il server comunicherà al client l’ID univoco della vaccinazione.

E’ inoltre presente il metodo openSocket(), che permette la creazione di un nuovo socket.

**OperatoriForm** : Nella seguente classe è previsto che l’operatore sanitario scelga un’operazione da eseguire tra le due disponibili, ovvero l’iscrizione di un nuovo centro vaccinale o la registrazione di un neo-vaccinato.



**package cittadini**

**Utente** : Tale classe contiene il costruttore per due differenti tipi di utente : uno utilizzato nella classe IscrizioneVaccinato e l’altro usato nella classe Registrazione. Oltre ai costruttori, la classe contiene vari metodi get e set per ricevere specifici dati riguardo una particolare istanza di Utente.

**Registrazione** : La classe è usata dal cittadino che vuole registrarsi per la prima volta in seguito alla vaccinazione. Vengono raccolti i dati relativi al cittadino, in particolare l’id univoco e il centro di vaccinazione, per poi essere inviati al server, che a sua volta li inserirà nel DB.

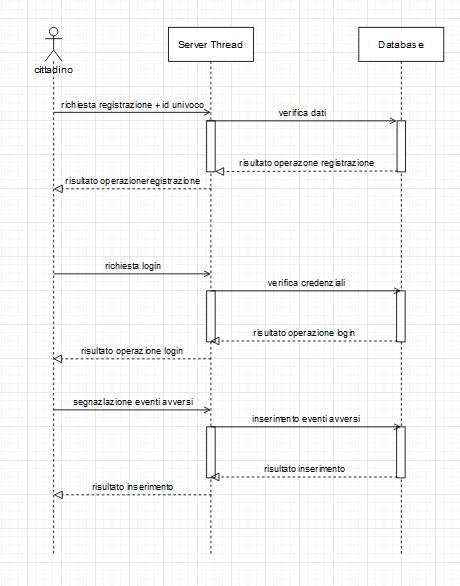
E’ inoltre presente il metodo openSocket(), che permette la creazione di un nuovo socket.

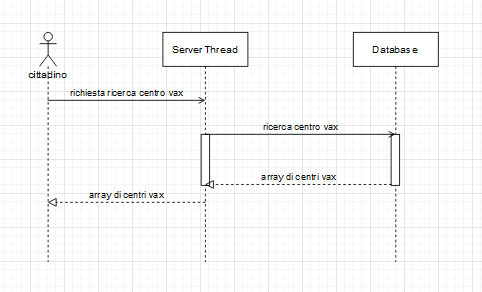
**AccessoAutenticato** : In questa classe è presente il metodo tramite cui il cittadino, precedentemente registratosi, ha la possibilità di segnalare i propri eventi avversi riscontrati dopo la vaccinazione. Tali dati vengono inseriti in una ArrayList di tipo Object e inviata al server, che a sua volta li inserirà nel DB. Essendo il numero di dati da inserire una costante ed essendo l’ArrayList di facile consultazione ci è parsa la scelta opportuna per il salvataggio di tali dati.

E’ inoltre presente il metodo openSocket(), che permette la creazione di un nuovo socket.

**CittadiniForm** : La classe offre al cittadino la possibilità di ricercare un centro vaccinale, o in base al nome parziale del centro vaccinale oppure in base al nome del comune e alla tipologia di centro (ospedaliero, hub, aziendale).

Oltre alla ricerca, l’utente avrà anche la possibilità di scegliere l’operazione di registrazione e/o login.





**SERVER**

Questo package è composto di una sola classe : **ServerThread**.

All’interno di questa classe come prima cosa è stato dichiarato il **ServerThread()** avente come unico parametro l’istanza di un nuovo socket, necessario affinchè ci sia parallelità nella risoluzione delle richieste del client. Sul socket è stato costruito un ObjectStream sia in input che in output. Non c’è stato bisogno di ulteriore astrazione rispetto all’Object poichè il compito di tradurli spetta al server, basandosi sul tipo di richiesta.

In seguito è stato dichiarato il **main**. Il primo passo da compiere all’interno del main è stato quello di verificare se fosse il primo accesso al server, per poter modificare e l’indirizzo IP e le credenziali di accesso al DB.

In seguito ci si connette al DB usando le variabili precedenti immesse e viene istanziato un nuovo socket, usato come parametro del nuovo ServerThread(s), che viene chiamato non appena viene ricevuta una nuova richiesta da parte del client, tramite Socket socket = s.accept.

Nel caso sia il primo avvio del DB vengono create le relazioni fondamentali al corretto funzionamento del DB ovvero “centrivaccinali”, “cittadini\_registrati” e la sequenza “IDprog” contenente i valori autoincrementali corrispondenti all’id univoco della vaccinazione da usare di volta in volta.

Immediatamente dopo il main è presente il metodo **connect()**, il quale restituisce un oggetto di tipo connection, ovvero una sessione tra il DB e il server fondamentale per la corretta assoluzione delle richieste SQL. Questo metodo è chiamato all’interno del main, dopo aver specificato le credenziali di accesso e indirizzo ip del server.

Dopo il precedente metodo verrà analizzato il fulcro del ServerThread, ovvero il metodo **run()**. Come prima azione viene letta la richiesta contenuta nell’oggetto di classe ConnessioneServer, ricevuta dal client tramite ObjectStream().

All’interno del run() è presente uno switch-case che si occupa di interpretare la richiesta e comportarsi di conseguenza.

* **case “centroVax”** : viene chiamato il metodo registraCentroVaccinale();
* **case "registrazioneVaccinato" :** viene chiamato il metodo registraVaccinato(), che oltre ad inserire i dati del vaccinato nel DB, restituisce l’id univoco della vaccinazione. Questo viene settato come variabile di ConnessioneServer attraverso il metodo setObj(id), il quale verrà rinviato al client, essendo un id necessario alle eventuali future azioni del cittadino;
* **case "registrazioneCittadino"** :viene chiamato il metodo registraCittadino();
* **case "srcCentroVax" :** come prima azione viene creato lo statement contenente la query per ricercare nel DB il centro vaccinale col nome comunicato dal client. L’introduzione del carattere ‘%’ , alla fine della stringa corrrispondente alla query, simboleggia la possibilità di essere interpretato come un qualsiasi insieme di caratteri. Ciò rende possibile la ricerca del centro per nome parziale.

Viene quindi chiamato il metodo cercaCentroVaccinale(), il quale risultato della query verrà poi inserito in una ArrayList di tipo CentroVaccinale. Questa struttura dati verrà poi restituita al client.

* **case "ricercaCVComuneTipologia"** : viene creata una stringa corrispondente alla query per ricercare il centro vaccinale in base al comune e alla tipologia di appartenenza del centro vaccinale. Il comune e la tipologia vengono comunicati dal client attraverso un array di string. In seguito viene chiamato il metodo cercaCentroVaccinale() passandogli come argomento lo statement basato sulla string query precedentemente creata. Il risultato del metodo viene inserito in una ArrayList di tipo CentroVaccinale. Questa struttura dati verrà poi restituita al client.
* **case "eventiAvversi":** si riceve dal client una ArrayList di tipo Object contenente gli eventi avversi comunicati dal client. Questa ArrayList viene passata come argomento del metodo inserisciEventiAvversi() insieme all’oggetto di tipo Connection.
* **case "LogIn":** si riceve dal client una HashMap di tipo String,String contenente lo userid e la password del cittadino, creati attraverso il processo di **registrazione**. Usare HashMap in questo caso ci è sembrato opportuno avente di default due campi, la chiave e il valore, che nel nostro caso verranno appunto usati per salvare userid e password. E’ inoltre di facile consultazione per i nostri scopi.

Viene quindi chiamato il metodo loginCittadino() passando come argomenti l’oggetto di tipo Connection e l’HashMap. La riuscita dell’operazione di login viene poi immessa nel primo campo dell’Hashmap, precedentemente svuotata, tramite una stringa contenente “vero” o “falso”. Dopo il precedente metodo viene chiamato getCentroVax(), passandogli come argomento l’ID del cittadino contenuto in HashMap, il risultato viene inserito nel secondo campo dell’HashMap.

L’HashMap è quindi stata riempita nuovamente coi nuovi valori e viene comunicata al client.

Veniamo quindi all’analisi dei metodi contenuti nel run() :

* **set\_get\_Id(Connection conn) :** viene inizializzata la variabile int id a 0, poi viene creato ed eseguito un PreparedStatement che seleziona il prossimo valore della sequenza auto-incrementale “idprog” . Il risultato della query viene estrapolato grazie al metodo next() del ResulSet. Il metodo **set\_get\_Id()** restituisce il valore della query, ovvero l’ID univoco

* **getCentroVax(Connection conn, String idUser) :** viene creato ed eseguito un PreparedStatement che seleziona il centro vaccinale di appartenenza di un utente identificato dal suo ID, passato al metodo proprio come argomento. Il risultato della query viene estrapolato grazie al metodo next() del ResulSet. Il metodo **getCentroVax()** restituisce il nome del centro vaccinale.
* **cercaCentroVaccinale(PreparedStatement statement)** : viene creata una ArrayList di tipo CentroVaccinale, un oggetto di tipo CentroVaccinale che è inizializzato come null e le variabili String nome, via, città, prov, tip e int nciv, CAP, severita\_media, n\_segnalazioni. Si crea quindi un ResultSet attraverso la quale si esegue la query contenuta nell’argomento statement di **cercaCentroVaccinale() .** Si prendono quindi i singoli valori di ogni Centro vaccinale risultante dalla query attraverso next() di ResultSet, che verranno poi passati come argomenti al costruttore di CentroVaccinale(). I centri vaccinali appena creati verranno inseriti di volta in volta nell’ ArrayList di tipo CentroVaccinale. Il metodo **cercaCentroVaccinale()** ritornerà proprio questa ArrayList.
* **registraVaccinato(Connection conn, Utente utente)**  : viene inizializzata la variabile int id a -1 . Poi viene creato uno statement che seleziona l’utente dalla tabella vaccinati\_”nomeCentroVax” e passato come argomento al metodo checkUserData(). Quest’ultimo metodo serve per verificare che l’utente non si sia già registrato. In caso esista già il suo record nella tabella viene comunicato la ridondanza dell’operazione di registrazione e viene restituito il suo ID. In caso non l’utente non sia già registrato, viene chiamato il metodo set\_get\_Id() che restituisce il prossimo valore di ID univoco. Successivamente si esegue un prepareStatement che inserirà i valori dell’utente nel DB, compreso l'id precedentemente ottenuto. Il metodo, anche **registraVaccinato()**  anche in questo caso restituisce l’id univoco.
* **registraCentroVaccinale(Connection conn, CentroVaccinale cv)** : viene creata una variabile Boolean inizializzata a true. Si esegue poi il metodo checkUserData() per verificare che il centro vaccinale passato come argomento del metodo princiale non sia già stato registrato. Nel caso in cui il record corrispondente al centrovaccinale in questione non esista già nel DB, viene creata dinamicamente la tabella vaccinati\_”nomeCentroVax” attraverso il metodo createTable(). Attraverso un preprareStatement e relativi metodi di setting, verranno inseriti i campi del centro vaccinale in questione nel DB. Il metodo ritorna un Boolean corrsipondente al successo o insuccesso dell’operazione.
* **registraCittadino(Connection conn, Utente user)** : viene creata una variabile Boolean inizializzata a false . Si esegue poi il metodo checkUserData() per verificare che l’utente passato come argomento del metodo princiale non sia già stato registrato. Nel caso in cui il record corrispondente all’utente in questione non esista già nel DB,attraverso un preprareStatement e relativi metodi di setting, verranno inseriti i campi dell’utente in questione nel DB.Il metodo ritorna un Boolean corrsipondente al successo o insuccesso dell’operazione.
* **loginCittadino(Connection conn,HashMap <String, String> datiLogIn)** : viene creato un PrepareStatement che seleziona userid e password, corrispondenti a quelli passati come argomento nell’HashMap, e viene eseguita la query. Attraverso il metodo next() di resultSet si verifica che ci sia un effettivo risultato dalla query. Se il risultato esiste, i valori contenuti nell’HashMap erano giusti e quindi l'operazione di login viene consentita. In caso contrario, i valori erano sbaglliati e l’operazione login viene negata. Il metodo restituisce una string contenente o “true” o “false”, simboleggiante la riuscita dell’operazione di login.
* **inserisciEventiAvversi(Connection conn, ArrayList<Object> Eventi\_Avversi)** : per prima cosa si ottiene il nome del centro vaccinale consultando il primo campo dell’ArrayList passata come argomento. Poi attraverso il metodo createTable() si crea la relazione "centro Vax"eventi avversi nel caso non esista già. Successivamente si crea uno statement che inserirà nella precedente tabella l’intensità degli eventi avversi e le note opzionali riguardanti questi ultimi. Tramite un ciclo for si consulta l’ArrayList Eventi\_Avversi per calcolare la somma della gravità di ogni singolo evento avverso (utile successivamente per eseguire la media) e inoltre, settare i valori delle intensità e relative note opzionali per la corretta esecuzione dell’inserimento nella tabella.

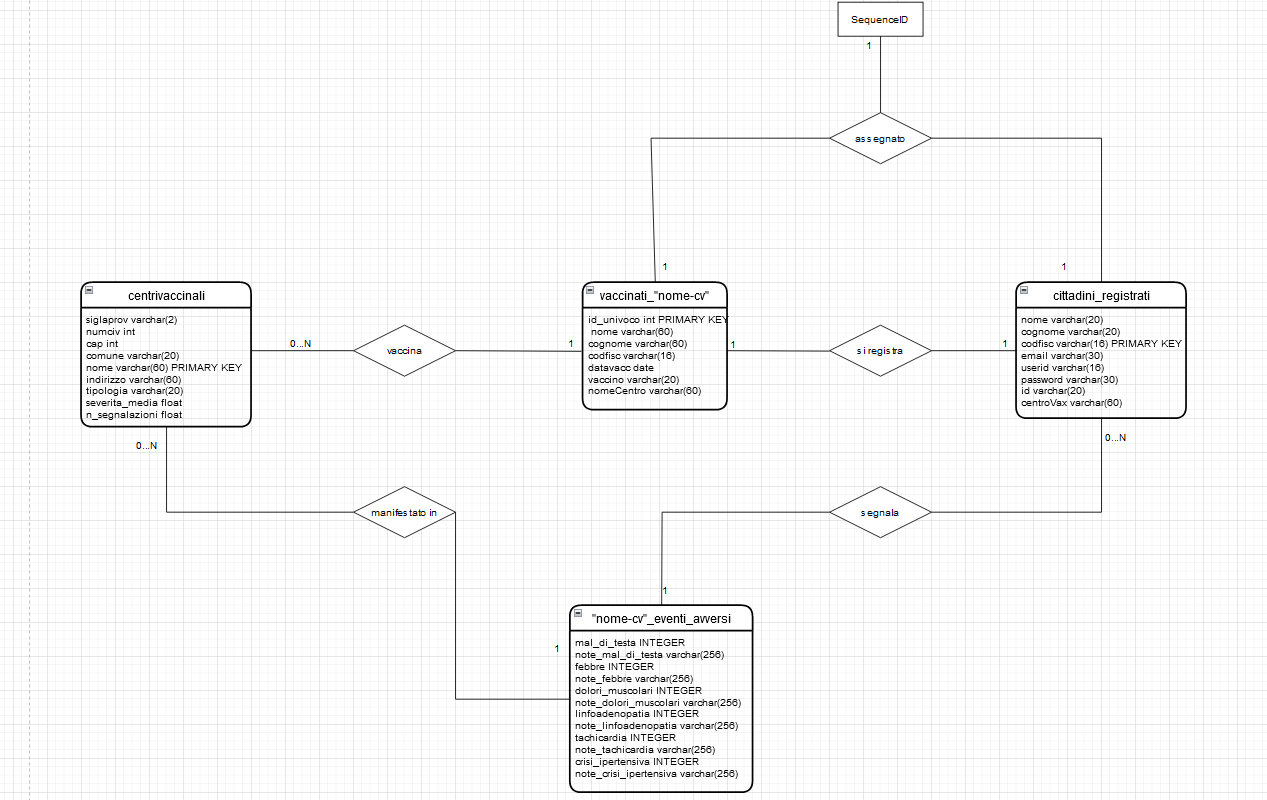
Si calcola la media dell’intensità degli eventi avversi e si chiama il metodo updateEventiAvversi() passandogli il nome del centro vaccinale e la media appena calcolata. Infine si esegue lo statement. Il metodo ritorna un Boolean indicante la corretta esecuzione del metodo.

* **updateEventiAvversi(String centroVax, float sev\_media\_att)** : si crea uno statement che servirà a selezionare la severità media e il numero di segnalazioni del centro vaccinale passato come argomento. Si esegue quindi la query. A questo punto si incrementa il numero di segnalazioni e si calcola la nuova media globale della severità degli eventi avversi usando questa formula : sev media tot = (sev media att+(sev Media Prec\*nEvE Prec))/nEvA. Ciò consente di non dover tenere registrate più colonne del necessario nella relazione “centrivaccinali”.

Infine si esegue un update di “centrivaccinali” coi valori appena calcolati. Ciò permette di tenere continuamente traccia della media globale degli eventi avversi e il numero di segnalazioni. Il metodo non ritorna nulla in quanto void.

* **checkUserData(Connection conn, PreparedStatement statement) :**  il metodo esegue la query passatagli come argomento. Se esiste la query da risultato il metodo ritorna true, altriementi false
* **createTable(Connection conn, String create\_table\_query) :**  viene creato uno PreparedStatement passandogli come argomento la stringa parametro del metodo principale. Esegue un update che creerà la tabella nel caso non esista già.

**DATABASE**

In seguito ad una attenta analisi dei requisiti ci è parso opportuno strutturare il database come segue :

* **tabella centrivaccinali** : questa tabella contiene le informazioni relative a tutte le istanze dei centri vaccinali che sono stati registrati. Sono anche presenti i campi rappresentanti la severità media e il numero di segnalazioni avvenute.
* **tabella cittadini\_registrati** : questa tabella contiene le informazioni relative a tutte le istanze dei cittadini registrati che hanno effettuato la registrazione dopo essere stati vaccinati.
* **tabella vaccinati\_”nome-cv”** : contiene le informazioni relative alle istanze dei singoli vaccinati per ogni centro vaccinale. Queste tabelle vengono create automaticamente ogni volta che un centro vaccinale viene registrato.
* **tabella “nome-cv”\_eventi\_avversi** : questa tabella contiene per ogni severità l'intensità riscontrata e una nota facoltativa a riguardo. E’ inoltre fondamentale il fatto che la tabella viene creata automaticamente alla prima segnalazione riguardante il centro vaccinale in questione.
* **sequenza sequenceID** : questa sequenza contiene i valori autoincrementali rappresentanti l’id univoco dei vaccinati.